INFORMAL TECHNICAL REPORT

For
SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS
(STARS)

*Organization Domain Modeling (ODM) Guidebook
Version 2.0*

STARS-VC-A025/001/00

14 June 1996

CONTRACT NO. F19628-93-C-0130

Prepared for:
Electronic Systems Center
Air Force Systems Command, USAF
Hanscom, AFB, MA 01731-2816

Prepared by:
Lockheed Martin Tactical Defense Systems
9255 Wellington Road
Manassas, VA 20110-4121

19970220 093

DTIC QUALITY INSPECTED 1

Data Reference: STARS-VC-A025/001/00
INFORMAL TECHNICAL REPORT
Organization Domain Modeling (ODM) Guidebook
Version 2.0

Distribution Statement "C"
per DoD Directive 5230.24
Authorized for public release; Distribution is unlimited.

Abstract

This guidebook describes the Organization Domain Modeling (ODM) domain engineering method. ODM has been developed by Mark Simos of Organon Motives to systematize key aspects of the domain modeling process and provide an overall framework for a domain engineering life cycle. ODM is intended as a highly tailorable and configurable domain engineering process model, useful for diverse organizations and domains, and amenable to integration with a variety of software engineering processes, methods, and implementation technologies. In addition to focusing on the strictly technical aspects of domain engineering, the method emphasizes the importance of clearly defining the organization context within which each domain engineering effort is conducted.

The primary purposes of the guidebook are to:

- Provide a definitive ODM reference document which promotes public understanding of the method and its applicability through in-depth descriptions of ODM concepts, processes, and workproducts.

- Provide substantial practical guidance for using the method by describing ODM activities in detail and offering workproduct templates and examples to get practitioners started.

- Provide general, high-level guidance in tailoring the method for application within a particular project or organization.

Data Reference: STARS-VC-A025/001/00
INFORMAL TECHNICAL REPORT
Organization Domain Modeling (ODM) Guidebook
Version 2.0

Principal Author(s):

Mark Simos, Organon Motives, Inc.	6/14/96 Date
Dick Creps, Lockheed Martin Tactical Defense Systems	6/14/96 Date
Carol Klingler, Lockheed Martin Tactical Defense Systems	6/14/96 Date
Larry Levine, Organon Motives, Inc.	Date
Dean Allemang, Organon Motives, Inc.	6/14/96 Date

Approvals:

Teri F. Payton, Lockheed Martin Tactical Defense Systems STARS Program Manager	6/14/96 Date

(Signatures on File)

REPORT DOCUMENTATION PAGE

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE 14 June 1996	3. REPORT TYPE AND DATES COVERED Informal Technical Data

4. TITLE AND SUBTITLE

Organization Domain Modeling (ODM) Guidebook, Version 2.0

5. FUNDING NUMBERS

F19628-93-C-0130

6. AUTHOR(S)

Mark Simos, Organon Motives; Dick Creps, Lockheed Martin; Carol Klingler, Lockheed Martin; Larry Levine, Organon Motives; Dean Allemang, Organon Motives

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Lockheed Martin Tactical Defense Systems
9255 Wellington Road
Manassas, VA 20110-4121

8. PERFORMING ORGANIZATION REPORT NUMBER

Document Number
STARS-VC-A025/001/00

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Department of the Air Force
ESC/ENS
Hanscom AFB, MA 01731-2816

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

A025

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Distribution "A"

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This guidebook describes the Organization Domain Modeling (ODM) domain engineering method. ODM has been developed by Mark Simos of Organon Motives to systematize key aspects of the domain modeling process and provide an overall framework for a domain engineering life cycle. ODM is intended as a highly tailorable and configurable domain engineering process model, useful for diverse organizations and domains, and amenable to integration with a variety of software engineering processes, methods, and implementation technologies. In addition to focusing on the strictly technical aspects of domain engineering, the method emphasizes the importance of clearly defining the organization context within which each domain engineering effort is conducted.

14. SUBJECT TERMS

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

Data Reference: STARS-VC-A025/001/00
INFORMAL TECHNICAL REPORT
Organization Domain Modeling (ODM) Guidebook
Version 2.0

Table of Contents

Data Reference: STARS-VC-A025/001/00
INFORMAL TECHNICAL REPORT
Organization Domain Modeling (ODM) Guidebook
Version 2.0

Table of Contents

Data Reference: STARS-VC-A025/001/00
INFORMAL TECHNICAL REPORT
Organization Domain Modeling (ODM) Guidebook
Version 2.0

Table of Contents

Data Reference: STARS-VC-A025/001/00
INFORMAL TECHNICAL REPORT
Organization Domain Modeling (ODM) Guidebook
Version 2.0

Table of Contents

Data Reference: STARS-VC-A025/001/00
INFORMAL TECHNICAL REPORT
Organization Domain Modeling (ODM) Guidebook
Version 2.0

List of Exhibits

Data Reference: STARS-VC-A025/001/00
INFORMAL TECHNICAL REPORT
Organization Domain Modeling (ODM) Guidebook
Version 2.0

List of Exhibits

Data Reference: STARS-VC-A025/001/00
INFORMAL TECHNICAL REPORT
Organization Domain Modeling (ODM) Guidebook
Version 2.0

List of Exhibits

Prologue

This document is version 2.0 of the *Organization Domain Modeling (ODM) Guidebook*. The guidebook describes the ODM domain engineering method developed principally by Mark Simos of Organon Motives, Inc., with sponsorship from the Software Technology for Adaptable, Reliable Systems (STARS) program (on behalf of the Defense Advanced Research Projects Agency (DARPA)) and the Hewlett-Packard Company.

The principal author of the guidebook was Mark Simos, with assistance from Dick Creps and Carol Klingler of Lockheed Martin Tactical Defense Systems and Dean Allemang and Larry Levine of Organon Motives.

This guidebook is a revision of version 1.0, published in March 1995. As before, the guidebook's central objective is to provide practical guidance for using ODM. It has been improved in a number of ways to better achieve those objectives, based on lessons learned and reader feedback. It describes the ODM process in greater detail than before, includes more workproduct examples, presents key concepts with greater clarity, and simplifies some of the terminology. The major changes are documented in more detail below.

The guidebook is still less complete and consistent than the authors would like. It would be useful to further clarify and refine the ODM process descriptions, incorporate an even more thorough set of workproduct examples and templates, and re-design portions of the document to further ease comprehension of the material. It would also be desirable to publish a companion volume offering more extensive practical guidelines for tailoring and applying ODM. No further revisions or volumes are planned as part of the STARS program, but the companies involved intend to continue to mature the ODM method and its supporting documentation in the future in response to ongoing lessons learned.

Changes from Version 1.0

The following general or pervasive changes were made to the document:

- The process descriptions have been supplemented with extensive example material, both in the form of sample workproducts and example items in running text. In most cases, the example material is presented as a single running example extending throughout the process. These examples serve to illustrate the process more clearly than in version 1.0.

- The process descriptions have been reorganized significantly to support greater explanation of the rationale underlying the activities and workproducts.

- The Core Concepts section has been expanded and clarified, and other conceptual material has been distributed throughout the document to provide more localized explanations of concepts relating to specific processes.

- A significant number of workproducts and all subworkproducts have been eliminated.

- Terminology has been simplified and clarified (we hope!).

The following significant high-level structural changes were made to the ODM process model:

- The *Orient Domain* task in the *Define Domain* sub-phase has been split into two new tasks: *Focus Domain* and *Situate Domain*.

- The *Acquire Domain Information* sub-phase has been reorganized. The latter two tasks have been changed from *Examine Artifacts* and *Elicit Informant Data* to *Elicit Data* and *Integrate Data*, respectively.

- The *Select Descriptive Model Types* task has been removed from the *Develop Descriptive Models* sub-phase, and the sub-phase has been renamed *Describe Domain*. The function of the *Select Descriptive Model Types* task has generally been reallocated to *Integrate Descriptive Models* in the *Refine Domain Model* sub-phase.

- The *Extend Domain Model* task in the *Refine Domain Model* sub-phase has been renamed *Resolve Domain Model*.

Changes at the more detailed levels of the process model are too numerous to list here.

Acknowledgments

For Feedback and Additional Information

We enthusiastically encourage trial use of ODM and solicit reader review and comments as input to the future evolution of the method and supporting materials. To learn more about ODM, discuss how to obtain support in applying it, or submit feedback on the guidebook and/or your practical ODM experiences, please submit e-mail to the following addresses:

> odm@organon.com
> odm@reston.unisysgsg.com

or contact the following individuals:

Mark Simos	Dick Creps
Organon Motives	Lockheed Martin
One Williston Road, Suite 4	9255 Wellington Road
Belmont, MA 02178	Manassas, VA 22110
Phone: (617) 484-3383 x11	Phone: (703) 367-1353
Fax: (617) 383-3363	Fax: (703) 367-1389
E-mail: mas@organon.com	E-mail: creps@reston.unisysgsg.com

Part I: Introduction

1.0 Guidebook Overview

This guidebook describes the Organization Domain Modeling (ODM) domain engineering method. ODM has been developed to systematize key aspects of the domain modeling process and provide an overall framework for a domain engineering life cycle. In addition to focusing on the strictly technical aspects of domain engineering, the method emphasizes the importance of clearly defining the organization context within which each domain engineering effort is conducted.

1.1 Purpose and Scope

The primary purposes of this guidebook are to:

- Provide a definitive ODM reference document which promotes public understanding of the method and its applicability through in-depth descriptions of ODM concepts, processes, and workproducts.

- Provide substantial practical guidance for using the method by describing ODM activities in detail and offering workproduct templates and examples to get practitioners started.

- Provide general, high-level guidance in tailoring the method for application within a particular project or organization.

The guidebook is **not** directly intended to do the following:

- Provide extensive justification and rationale for reuse and domain engineering in general or for the ODM domain engineering approach in particular.

- Define the specific role of ODM within an overall software engineering process or life cycle.

- Prescribe a full range of specific methods that can be used to support domain engineering, beyond those that are considered to be inherent or unique to the "core" ODM method, as described herein. (However, the guidebook does identify several general categories of supporting methods and describes where they are applicable within ODM.)

- Prescribe specific tools that can be used to support ODM. (However, in conjunction with the supporting methods, some categories of supporting tools are identified.)

- Compare ODM with other domain analysis, domain engineering, or reuse methods or processes (although it may be useful in helping the reader to draw such comparisons).

All of these are potential topics for future papers and reports that will complement the guidebook. In particular, ODM's relationship to the overall software life cycle and to specific techniques and tools will vary from one organization to another. These issues thus contribute to the general problem of tailoring ODM to specific organization contexts. Although this guidebook provides general tailoring guidance, there is a need for a companion document that provides more extensive guidance in tailoring ODM.

1.2 Audience

This guidebook is targeted to readers having one or more of the following organizational roles:

- *Domain Engineer* — Responsible for scoping and modeling domains of interest to an organization and designing and implementing collections of assets ("asset bases") that can be reused in multiple application systems. Interested in learning to apply domain engineering concepts, processes, methods, and tools.

- *Program/Project Planner* — Responsible for planning the objectives, strategy, processes, infrastructure, and resources for software engineering programs or projects. Interested in incorporating systematic, domain-specific reuse into those programs/projects.

- *Reuse Advocate* — Responsible for keeping abreast of reuse concepts, technology, and trends and promoting the establishment/improvement of reuse capabilities and practices within an organization. Interested in understanding how new concepts, processes, methods, and tools can be applied to accelerate reuse adoption.

- *Process Engineer* — Responsible for defining, instantiating, tailoring, installing, automating, monitoring, administering, and evolving software engineering process models. Interested in defining reuse processes or integrating them with overall life cycle process models.

1.3 Benefits

By reading this guidebook, segments of the audience should be better able to do some or all of the following:

- Understand key ODM concepts, including:

  - The ODM process flow from Plan Domain --> Model Domain --> Engineer Asset Base, the reasoning for this flow, and the key process steps involved

  - The need to model organization context as an integral part of domain engineering

  - The difference between system and domain modeling techniques and the distinct roles these techniques play in domain engineering

- Assess and communicate the benefits and implications of applying ODM concepts, e.g.:

  - Business problems and opportunities where domain modeling may be useful

  - Value of ODM methods and processes in relation to other software engineering methods, processes, and tools

  - Ways ODM can be tailored and integrated to meet an organization's needs

- Decide whether or not to explore ODM further

When supplemented with further reading, training, and consultation, the guidebook should enable readers to:

- Assess their organization's receptivity to applying ODM and related methods

- Initiate, plan, manage, and perform a domain engineering project using ODM

- Make technology choices for the various supporting methods required to perform a complete ODM domain engineering project

1.4 Guidebook Organization

The body of the guidebook is organized into three major parts:

- **Part I: Introduction**

    — *Section 1: Document Overview (this section)* — Defines the purpose, scope, audience, benefits, and organization of the document and offers guidance in how to read it.

    — *Section 2: ODM Background* — Describes the factors that motivated the development of ODM, the context in which it was developed, how it has been applied, and how it relates to other reuse-oriented work.

    — *Section 3: ODM Core Concepts* — Articulates the conceptual foundations of ODM in terms of distinctive features of ODM in meeting reuse and domain engineering needs.

- **Part II: ODM Processes and Products**

    — *Section 4: Domain Engineering Life Cycle* — Describes the overall ODM process in terms of a domain engineering life cycle model.

    — *Section 5: Plan Domain* — Describes the ODM processes involved in planning a domain engineering project, including scoping and defining the domain of focus.

    — *Section 6: Model Domain* — Describes the ODM processes involved in modeling the commonality and variability of concepts and features in the domain.

    — *Section 7: Engineer Asset Base* — Describes the ODM processes involved in selecting the specific domain features to be supported in a reusable asset base and architecting and implementing the asset base.

- **Part III: Tailoring and Applying ODM**

    — *Section 8: Supporting Methods* — Describes several categories of methods and techniques (external but complementary to ODM) that can be applied to support aspects of the ODM process.

    — *Section 9: Optional Layers* — Describes several process layers that can optionally be added to ODM to provide capabilities that may be needed in some project contexts.

    — *Section 10: Guidelines for Applying ODM* — Provides a set of basic, practical guidelines for tailoring and applying ODM on a domain engineering project.

The guidebook also includes the following supplementary material:

- **Appendix A: ODM Process Model** — A graphical view of the ODM process model, expressed in the $IDEF_0$ notation. (The $IDEF_0$ diagrams are also embedded within the process descriptions in Part II.)

- **Appendix B: ODM Lexicon** — Definitions of the terms that are fundamental to understanding ODM. Includes descriptions of ODM workproducts.

- **Appendix C: ODM Workproducts** — A catalog of the ODM workproducts and a set of

3

templates for worksheets that support development of selected workproducts.

* **References** — Bibliographic entries for all documents referenced in the guidebook.

1.5 How to Read the Guidebook

Each segment of the audience has different needs and interests and will thus benefit most from different portions of the guidebook. All segments of the audience should read Part I (in conjunction with the ODM lexicon in Appendix B) to gain a basic understanding of the method. The Program/Project Planner and the Process Engineer should read Part III to gain general insight into how the method can be tailored and applied. The Process Engineer and the Domain Engineer should read Part II and the appendices to learn about the ODM processes and work products in detail, but from differing perspectives: the Process Engineer to gain insight into tailoring and integrating the ODM process to support domain engineering, and the Domain Engineer to learn how the process can be enacted to produce reusable domain assets.

Although the guidebook can be read sequentially from beginning to end, it has been structured to support alternative reading styles. As indicated above, Part I should be read first, but Parts II and III can be read more or less independently. Furthermore, the sections within Parts II and III need not be read in a strictly sequential order. This is due in part to the fact that the guidebook was developed in accordance with the Unisys STARS Process Definition Process [Klin95a], which imposes a clear discipline for structuring and presenting process information. The structure of Part II mirrors the hierarchical structure of the ODM process model, as depicted in graphical process trees and IDEF$_0$ diagrams. This structure makes it easy to find and read descriptions of specific portions of the process model. In addition, the sections in Part II have a well-defined internal structure that enables them to be read and understood relatively easily without a global understanding of the process. In practice, however, it is useful to read about a process in conjunction with the processes that surround it, including its parent processes. Further guidance for how to read and interpret the process descriptions is provided in an introductory portion of Part II.

Another important aspect of the guidebook structure is the treatment of key ODM terms and workproducts. In general, when a key term is first introduced in the guidebook, it appears in a *bold italic* typeface and is defined at that point. Such terms are also sometimes shown in bold italic when first appearing within sections of the document to reestablish their identity as lexicon terms. All such terms also appear, with definitions, in the ODM lexicon in Appendix B so they can be easily looked up whenever they are encountered in the document.

The ODM workproducts and data items (all of which are represented as data flows in the IDEF$_0$ diagrams) appear in a SMALL CAPS typeface wherever they are referenced in the document. Descriptions of the workproducts appear both in the descriptions of the processes that create them and in the lexicon appendix. Appendix C includes a workproduct catalog listing the workproducts alphabetically and in terms of the processes which create them. The latter list references the sections in which the processes are described, thus providing an additional navigational aid. Appendix C also includes templates for worksheets to aid in developing selected workproducts.

To further ease reading and navigation of the guidebook, the title and number (or letter) of the current section or appendix appears in the header atop each page.

NOTE: Because ODM is closely related to concepts in the STARS Conceptual Framework for Reuse Processes (CFRP) model, we recommend that the reader become familiar with the CFRP by reading either the CFRP Description document [CFRP93a] or a paper on the CFRP [Crep95a] prior to reading the remainder of the guidebook (particularly Parts II and III). See Section 2.5 and Section 4.0 for more details on ODM's relationship to the CFRP.

4

2.0 ODM Background

2.1 Origins

Organization Domain Modeling (ODM) is a systematic approach to domain engineering developed by Mark Simos of Organon Motives over the past eight years. The ODM approach originated during an early Unisys STARS project to develop the Reuse Library Framework (RLF), a hybrid semantic network and rule-based knowledge representation system intended for use as a domain modeling tool for software reuse [Unis88a, Sold89a]. (Mark Simos was initial technical lead on this project while with Unisys.) The RLF was developed by conducting a domain analysis of knowledge representation systems used in the expert systems field and producing a layered architecture and implementation based on the results of the analysis. The domain analysis approach developed on this project included many of the core ODM ideas and established an initial foundation for the ODM method.

Subsequently, considerable support and collaboration in refining ODM has come from the Hewlett-Packard Company and from Lockheed Martin Corporation (formerly Loral Defense Systems-East and Unisys Government Systems Group), as a STARS prime contractor. In particular, the STARS Program has devoted significant resources over the past four years to refining ODM and defining its relationships with broader reuse process models and supporting tools. This is now culminating in the Lockheed Martin STARS Reuse Whole Product, in which ODM plays a central role (see Section 2.5 below).

The scope of the method has been further broadened through reuse consulting work with various organizations [Simo91b] and by synthesizing other research in domain analysis and domain engineering [Prie91a, ADER96a] with work in non-software engineering disciplines such as organization redesign and workplace ethnography [e.g., Seng90, Seng94]. As a result, ODM emphasizes the organizational and non-technical issues associated with domain engineering more than comparable methods, while retaining a strong technical foundation rooted in a variety of technical disciplines.

2.2 Motivation

ODM was developed to address a gap in the domain engineering methods and processes available to reuse practitioners. Because work in domain analysis for software reuse emerged out of the software engineering research community, many of the methods were developed by technologists. Domain analysis was seen as primarily a technical modeling problem, often little more than a set of extensions to system modeling techniques to accommodate representations of variability. The resulting body of techniques range from informal to formal in terms of representations but are often quite *ad hoc* in terms of actual processes followed. More importantly, they leave unaddressed some of the most difficult problems would-be domain engineers face in trying to launch real domain engineering projects in business settings.

On the other hand, there has been a great deal of work done on the non-technical aspects of reuse adoption, including economic cost models for reuse, technology transfer strategies, and broad organizational models for structuring reuse groups within software engineering organizations. The problem with this work as a starting point for domain engineering is that it generally assumes massive changes on the part of organizations; these depend in turn on a high level of commitment from both top-level management and from the engineering trenches. In most practical settings, such commitment comes only after some pilot projects have demonstrated the viability of a reuse-based approach within that organization.

People trying to launch small-scale domain engineering projects have to negotiate the full range of technical and non-technical issues in planning, managing, and transitioning their work. Yet they typically will not have the support of an already existing reuse infrastructure within the organization. In fact, motivating organizations to adopt reuse practices on a more global scale may well depend on successful results from initial pilot projects. There is a pressing need for more detailed guidance on how to define and manage these initial domain engineering pilot projects. The ODM method is an attempt to meet this need and establish a foundation for evolving initial pilot projects into thriving reuse programs.

2.3 Objectives

Specific objectives of ODM include:

1) Make the domain engineering process more systematic, formal, manageable and repeatable. In particular, document where key boundary negotiation and scoping activities take place, to avoid the phenomenon of *ad hoc* designs being imposed as domain models.

2) Ground domain engineering projects in a specific organization context. Support a view of modeling activities as not only describing the state of the organization, but also directly contributing to its evolution.

3) Maximize use of legacy artifacts and knowledge as:

   — an empirical basis for systematic scoping of domain definitions, models and asset bases.

   — a source of domain knowledge.

   — potential resources to use in reengineering.

4) Reveal the hidden constraints embedded in legacy systems and artifacts and render them visible in domain models and assets.

5) Encourage exploration of maximum variability within the domain, including opportunities in various dimensions:

   — Reuse of artifacts from across the software life cycle.

   — Reuse of process as well as product assets.

   — Reuse in ways that facilitate use of static components and generative techniques, as well as hybrid strategies for asset implementation.

   — Reuse via definition of flexible, highly developed architectures for the asset base.

6) Provide effective strategies for selecting an intended scope of applicability for asset bases that is strategically appropriate for the performing organizations. In particular, apply techniques that limit the impact of combinatorial sets of choices in selecting range of variability.

7) Support evolution of the asset base, and the scale-up of the technology to support new kinds of organizations, organized around domains rather than around systems or products in a conventional sense.

Many of these objectives are mutually supportive. For example, systematizing the process (Objective 1) is closely related to effective scoping strategies (Objective 6); however, there are

additional elements involved in systematizing the process; and systematizing the process is not the only reason for scoping strategies.

2.4 Scope and Applicability

ODM is intended as a highly tailorable and configurable domain engineering process model, useful for diverse organizations and domains, and amenable to integration with a variety of software engineering processes, methods, and implementation technologies. In terms of the STARS Conceptual Framework for Reuse Processes (CFRP) [CFRP93a], the core ODM method specifically addresses processes involving the creation of reusable assets, as well as elements of reuse planning and infrastructure development. ODM is thus directly applicable to planning and performing domain engineering projects in the context of an overall reuse program. (See Section 2.5 below and Section 4.0 for more information about ODM's relationship to the CFRP.)

Where ODM Can Be Applied

ODM was developed primarily to support domain engineering projects for domains which are:

- mature (i.e., a set of legacy systems exists)

- reasonably stable (i.e., at least some of the legacy systems are worth examining)

- economically viable (i.e., new systems are anticipated in the domain).

ODM may be applicable in circumstances where not all these criteria are satisfied, but ODM is most applicable when they are all met. Also, since ODM has been developed in collaboration with both government and commercial software organizations, it is applicable to organizations and domains in either of these contexts.

ODM can be applied on domain engineering projects of arbitrary scope, in the context of reuse programs that are just starting or are very mature. It is recommended that organizations just beginning a reuse program apply ODM on a pilot domain engineering project focusing on a relatively small domain. This initial foundation can then be evolved into a broader reuse program.

Where ODM May Not Be Applicable

In terms of the CFRP, the core ODM method does not directly address the ongoing management of domain models and assets, nor utilization of the assets by application development projects. ODM also does not encompass overall reuse program planning, including the establishment of producer-consumer relationships between domain engineering projects and other efforts, such as system reengineering projects or planned new products.

ODM may not be applicable within organizations that are not prepared to commit to, or at least experiment with, the notions of systematic reuse inherent in the CFRP and ODM. Also, not all organizations may be prepared to adopt the level of modeling rigor or the modeling styles or approaches recommended within ODM. Although ODM has been designed for integration with a wide range of supporting methods and tools, some organizations may not currently possess a technology infrastructure and level of technical expertise sufficient to support ODM modeling needs. See Section 10 to gain more insight into whether ODM is adaptable to your organization's needs.

2.5 Relationship to Other Products

Within the STARS program, ODM is being developed and refined as part of the Lockheed Martin STARS *Reuse Whole Product*, inspired by Geoffrey Moore's "Crossing the Chasm" model of technology transition [Moor91a]. The Reuse Whole Product includes a set of reuse support technologies that the Lockheed Martin Tactical Defense Systems STARS team has developed, integrated, or used. As part of the Reuse Whole Product effort, these technologies are being further integrated and augmented with a comprehensive set of training materials and examples to unify the technologies and make them easier for practitioners to use in concert. The Reuse Whole Product has evolved into a strategic alliance involving Lockheed Martin Corporation, Organon Motives, Inc., WPL Laboratories, Inc., and Knowledge Evolution, Inc.

The Reuse Whole Product component technologies can be viewed as addressing reuse at differing levels, or layers, of abstraction. These layers, from highest to lowest level of abstraction, are termed *Concepts*, *Processes*, *Methods*, and *Tools*. In general, technology choices made at any layer will constrain the choices available at the layers below. The Reuse Whole Product reflects a specific set of technology choices made at each layer, as shown in Exhibit 1.

Level of Abstraction	Products
Concepts	STARS Vision Organon Vision CFRP
Processes	CFRP ROSE
Methods	ODM(& supporting methods) LIBRA
Tools	RLF/Open RLF KAPTUR ReEngineer

Exhibit 1. STARS Reuse Whole Product Technology Layers

The technical concepts framing the Reuse Whole Product stem from the STARS product line vision (originally called *megaprogramming*), which integrates the concepts of process-driven, domain-specific reuse-based software engineering, supported by modern tools and environments [Boeh92a, Fore92a]. This vision is augmented in the Reuse Whole Product by the *organon* concept, which is founded on the notion of repositories of codified domain knowledge, coupled with proactive technologies to support the use and evolution of the knowledge [Simo91a].

Within this context, the Reuse Whole Product is scoped specifically to provide integrated process and tool support for the ODM domain engineering life cycle. In addition to ODM, the major technologies in the Reuse Whole Product are:

- *STARS Conceptual Framework for Reuse Processes (CFRP)* — The CFRP is a consensus STARS product that provides a conceptual foundation and framework for understanding domain-specific reuse in terms of the processes involved. [CFRP93a, CFRP93b, Crep95a]

- *Reuse-Oriented Software Evolution (ROSE)* process model — A CFRP-based life cycle process model that partitions software development into Domain Engineering, Asset Management, and Application Engineering and emphasizes the role of reuse in software evolution.

[ROSE93a]

- *Learning and Inquiry Based Reuse Adoption (LIBRA)* approach — A multi-disciplinary set of techniques and guidelines for uncovering, and facilitating shifts in, the beliefs and behavior patterns that form core barriers to reuse adoption within an organization. [LIBR96]

- *Reuse Library Framework (RLF)* domain modeling toolset — A toolset developed originally by Unisys (now Lockheed Martin) and STARS which supports taxonomic domain modeling via semantic network and rule-based formalisms, and features graphical and outline-based model browsers [Sold89a]. Open RLF, a reimplementation of RLF to support browsing and editing of models via World Wide Web and CORBA interfaces, is currently being developed jointly by Organon Motives and WPL Laboratories.

- *Capture* domain modeling and legacy management toolset — A toolset developed originally by CTA and NASA (now in the process of transitioning to Knowledge Evolution, Inc.) which graphically supports comparative modeling of system artifacts and domain assets. [Bail92a]

- *ReEngineer* reengineering toolset — A toolset developed by Lockheed Martin to support the reengineering of legacy systems via fine-grained analysis and abstraction of system structure.

The Reuse Whole Product effort as funded by STARS will continue through late 1996. Several evolving products and supporting materials will be released and available for trial use during that period (including this guidebook and associated tutorial material).

Although the above technology choices are sound in the Reuse Whole Product context, ODM can be applied in conjunction with a wide variety of other methods and tools that support the domain engineering life cycle. In general, this guidebook has been written to be as independent of specific support technology choices as possible. Although a catalog of potential ODM support technologies is beyond the scope of this guidebook, Section 8 describes several categories of methods (and related tools) that can support ODM. Please consult that section for more details.

2.6 Applications To Date

ODM has been applied on a small scale by a variety of organizations, including Lockheed Martin Corporation, the Air Force Comprehensive Approach to Reusable Defense Software (CARDS) program, the Software Engineering Institute, Logicon Corporation (on behalf of the U.S. Navy Program Executive Office for Cruise Missiles and Unmanned Aerial Vehicles), and the Rolls-Royce University Technology Centre [Kell96]. In addition, the following relatively major ODM application efforts have been undertaken and produced good results:

- **Hewlett-Packard.** Some aspects of the ODM approach were evolved in working with HP application teams in RADAR (Reusability Analysis/Domain Analysis Review) sessions [Coll91a]. These were design reviews focusing on projects that have followed a system design or architectural modeling process similar to domain analysis, whether or not a formal domain analysis method, *per se*, was followed.

  The ODM process model was used as a basis for question sets use in the sessions to probe engineering teams to reflect both about the products and processes of domain analysis. The sessions served both as training for engineers in the concepts and methods of domain modeling, and to rapidly reveal the gaps and breakdowns in the method. Patricia Collins (HP Corporate Engineering) has since evolved these question sets into a domain engineering workbook specifically tailored to HP organizational objectives and business culture. This workbook is now being applied on numerous domain engineering efforts within HP divisions.

- **Army STARS Demonstration Project.** STARS worked with the Army, Navy, and Air Force
  to sponsor three DoD software engineering projects (termed the STARS "Demonstration
  Projects") from late 1992 through early 1996 to assess the STARS product line approach in
  realistic and familiar contexts. Lockheed Martin Tactical Defense Systems supported the
  Army STARS Demonstration Project, which was performed by the US Army Communica-
  tions and Electronics Command (CECOM) Software Engineering Directorate. The project
  focus was on domain engineering and system reengineering in a system maintenance context.

  ODM, in conjunction with other Reuse Whole Product technologies (particularly the CFRP
  and RLF), formed the basis for the domain engineering approach that was developed and
  applied on the project [ADER96a, Wick94a]. The experiences and feedback from adapting
  ODM for use on the project have substantially influenced the evolution of ODM into its cur-
  rent form. In addition, the project's experiences in applying RLF and other domain engineer-
  ing support technologies have shed considerable light on ODM tool support strategies in
  general. The project significantly formalized its domain engineering process and developed
  an initial Domain Engineering Guidebook [DEGB95a] describing a process to recommend
  for use on future projects.

  In 1995, the project was awarded a Federal Technology Leadership Award in recognition of
  its commitment to and successful application of new, ground-breaking technologies.
  Although many factors contributed to the project receiving this honor, the award is, at least in
  part, a tribute to and validation of the ODM method.

3.0 ODM Core Concepts

ODM builds on concepts that have emerged from the work of many researchers and practitioners in the field of "systematic software reuse." It is beyond the scope of this document to give a thorough introduction to this foundational work, which has been well documented in a number of readily available proceedings and collections of papers, including [ICSR96, SPC93a, SPC93b, Prie91c, Fore92a, Boeh92a]. However, it is necessary to establish a frame of reference for understanding the key differentiators of ODM.

To provide this frame of reference, this section provides the starting point a "core concepts roadmap" for the ODM method, introducing themes that reappear at varying levels of detail within the process descriptions in Part II of this document. For readers less familiar with the field of reuse, this may involve introduction to some new ideas. For readers more familiar with the reuse field, this may involve drawing distinctions between the ODM approach and some notions that are part of reuse "conventional wisdom."

The organization of this section is as follows:

- Section 3.1 begins from the perspective of the individual engineer's experience, and attempts to provide some background for the current state of practice in systematic reuse and domain engineering,

- Section 3.2 presents essential aspects of the engineering situations we would characterize as domain engineering, and offers some examples of situations that match these criteria.

- Section 3.3 describes some of the key challenges in domain engineering, and indicate reasons why conventional software engineering methods do not adequately address these challenges. We also identify some elements of reuse "conventional wisdom" which ODM refines or recharacterizes in critical ways, including the notion of a "domain".

- This leads to a discussion, in Section 3.4, of some of the key features of the ODM method as they respond to these challenges.

3.1 Domain Engineering Overview

The reusability of a software component is the quality of being predictably useful in new applications, including those not anticipated by the original developers. This is almost a paradoxical "requirement" that a component exceed its requirements. Software engineers often try to address reusability with various *ad hoc* approaches adapted from conventional software engineering techniques, which are primarily oriented towards a fixed problem for which alternative solutions are considered by the developer. These techniques break down in situations where the problem scope is itself a variable to be determined as part of the engineering process.

There are many informal examples of these kinds of situations, such as developing libraries of subroutines, application-specific languages, or object-oriented frameworks. None of these design problems are system or application engineering in the ordinary sense. They are more similar to product engineering tasks, with, however, some important distinctions.

The systematic reuse community has evolved the discipline known as domain engineering in an attempt to formalize these processes and make the design of reusable software more repeatable, verifiable and methodical. The goal of domain engineering can broadly be described as optimizing the software development process over a "window" of multiple applications that address a common business or problem domain. The systematic reuse community has established ambitious

11

goals for what a domain engineering process should address, such as the ability to support design for reuse for all products and processes of the software life cycle, not just source code.

But to date most methods for domain analysis have fallen short of these goals. They are limited by their assumptions of the types of organizational contexts in which domain engineering will be performed and the types of domains that can be addressed. In addition, because these methods were primarily developed by software engineers they embed assumptions about the kinds of software engineering methods and representations to be used in *domain modeling*, and have also failed to isolate the unique methodological challenges in domain engineering that are *not* adequately addressed by simple extensions to software engineering methods. ODM is, in effect, a "second-generation" domain engineering method that systematically addresses these issues.

The Reuse Paradox

Most software engineers share an informal intuition about what "reusability" ought to mean: you build a component with some specific set of uses in mind; over time, people find it useful in new ways, perhaps even ways the developers did not anticipate. Components like these can be compared to a good hand-tool like a woodsman's ax: they are simple, elegantly crafted with attention to detail, intuitive and natural to use, and versatile in application in the hands of a skilled worker.

Most of us have also known the frustrating experience of trying to reuse many software applications or components written today: rather than a continual discovery of new uses, it is a continual discovery of new problems. Contextual assumptions made by developers emerge to snag us unawares at every turn. The component almost seems to have a will of its own, always including either a "little too little" (or, just as often, a "little too much") functionality to serve the task at hand. Components like these are like the multi-function kitchen appliances ("not available in stores"--and for good reason!) that do ten jobs equally poorly.

So, informally and intuitively, we can usually tell the difference between components that possess or lack this quality of "reusability." But how do we capture this intuition in practical guidelines for building reusable software? In particular, how do we capture the requirement that a component should behave well in unanticipated new contexts of use? This seems to present a paradox: almost by definition, we are "requiring" that a component exceed its requirements, asking for a form of quality that can't be defined up front.

A central thesis of the ODM approach is that this seemingly ineffable requirement can be formalized into a distinct class of engineering problem; further, that such problems are amenable to a systematic design approach that addresses their unique methodological issues; and that this design approach is complementary to but distinct from the wide variety of methods for engineering software solutions for single contexts of application.

Ad Hoc Approaches

The first step is recognizing that there are aspects to such problems not adequately addressed by standard software engineering techniques. This turns out to be a difficult conceptual hurdle: because the task of designing software to be "reusable" looks suspiciously like other software development tasks on the surface, software engineers tend to apply techniques that have worked for them in other situations. Unlike an obvious new technology, reuse feels like something we ought to be able to do already, if we just are able to divine the "right abstraction." There are a few classic approaches followed:

- Reuse by fiat. As engineers, we sometimes pick our favorite component and simply declare it

as reusable or generic. We thus arrive at an abstraction based on a familiar example (e.g., a system we have built) and get excited by the commonality that is revealed. "All X systems," we announce, "are really nothing more than... [fill in the blank]!"

In the systematic reuse community, "leveraged" reuse usually refers to using a software work product originally developed for a single context in a new context, making whatever modifications are necessary to adapt it to the new requirements. The attitude described above goes a step farther by making a claim for broader reusability for the component in question. In either case, however, the problem is the hidden contextual assumptions that are brought along with the component, that interact in unpredictable and even dangerous ways in the new situation.

- Genericize. Often people try to achieve reuse by making a component "generic," which usually means "abstracting" or removing detailed (e.g., implementation) information.

  The problems with this approach are twofold. First, by eliminating the specificity much of the value added of reusing the component is reduced. At a certain point, all programs can be built with a "reusable" Turing machine, but this doesn't help us much. Second, even the high-level abstraction embeds choices and decisions that constrain the component's range of use. These choices and the scope of use they imply may become even more obscured through the "generic" process than in an individual component built for a specific application.

- Apply "universal" design principles. Reengineer the component, applying certain general or absolute principles of design claimed to produce better or more "reusable" software. Following this particular version of the Grail Quest, people vociferously advocate some design approach which matches their theories about universal design principles, principles they believe will "always" result in more reusable software.

  In fact, the principles in question are usually useful in a broad but not infinite range of contexts. Once again, this contextual range is sometimes invisible to the designer.

All these approaches break down because engineers have failed to grasp that they are facing a qualitatively different kind of problem in designing for reuse. Most software engineering methods implicitly or explicitly assume a single-application "problem-solving" paradigm. Whether the techniques are top-down (e.g., structured decomposition) or bottom-up (e.g., modular composition of objects) the design process is well-specified only to the extent that there is a customer set of needs to be identified and satisfied.

But developing for reuse is not developing to solve a single problem. (In fact, we will see that even the use of the word "problem" here is problematic!) There is something hard about the problem of designing for reuse which has nothing to do with how good we are at software engineering. Software engineers are trained to deal with problems in a certain way. It is as if having a "problem to be solved" created a sort of "gravity" for the engineer. When lifted out of the structuring confines of a *system engineering* problem, they suddenly must learn "juggling in free fall" (where "free fall" refers to being beyond the gravity of specific system requirements) [Simo94a]. In such circumstances, many competent engineers have watched the pins float from their hands.

3.2 Characteristics of a Domain Engineering Problem

The systematic reuse community has introduced the concept of *domain engineering* in an attempt to formalize the process of designing for reuse and making it more repeatable, verifiable and methodical. Domain engineering, while related to application or single-system engineering, has, at minimum, these distinct aspects:

- Multiple System Scope. System engineering involves designing the best solution for a single

intended application. Domain engineering involves designing assets that will optimize the overall engineering process for some set or class of multiple applications. Unlike developing a single application, where there is at least in principle some optimum (or at least adequate) single solution to be designed, in domain engineering we are often trying to model the *space of solution alternatives* themselves.

- Smaller than System Scope. System engineering, presented with a single application context, implicitly has the job of designing the "whole solution." Domain engineering, while spanning multiple applications, has the "luxury" of picking what scope of functionality to address across those systems. For example, a domain engineering effort addressing the family of custom applications created within a telecommunications company could focus specifically on the software implementing a specified class of *features* (e.g., call forwarding, call waiting) within those overall systems.

This means the domain engineer has several scoping decisions that are not part of the ordinary application development process: scoping the "range of coverage" for the components to be developed, both in terms of the number of systems within which the components will be applicable (the "market") and the specific features of those systems to be addressed by the components (the "coverage"). If these and other related methodological issues are not dealt with intentionally in the process followed, they can compromise what would otherwise be workable software engineering results. These basic concepts therefore turn out to have pervasive impact on all the phases, activities and workproducts of the domain engineering life cycle.

Informal Examples of Domain Engineering

By articulating the above criteria for what kinds of situations constitute a domain engineering "problem," it immediately becomes evident that software engineers continually bump up against such problems, even though they rarely call this domain engineering. Some examples of challenges (i.e., both "problems" and "opportunities") that could be considered forms of domain engineering include the following:

- Development of a library of standard routines for programmers for a given operating system (or language, or user interface) platform;

- Development of an object-oriented framework for a given class of applications;

- Definition and implementation of an application-specific language or scripting mechanism for performing certain well-scoped system tasks (e.g., makefiles, specialized typesetting languages, 4GLs);

- Creation of a standard template library for producing and managing certain categories of documents;

- Doing competitive analysis on the features of a class of commercially available software products, in order to identify both "best of breed" and major *variants*;

- Definition of a public interchange standard for some class of data (e.g., IDL specifications for classes of services);

- Developing a palette of primitive operations supported by a pull-down menu in a tool's interface;

- Defining a common software platform within or across a product line internal to an organization.

Clearly, none of these design tasks are equivalent to building a single "application" in the conventional sense.

The Notion of an Asset Base

If domain engineering involves a scope that can simultaneously address multiple systems, and only a subset of the functionality of those systems, what exactly are we engineering when we do domain engineering? The most familiar notion to software engineers would be the idea of a "component": a piece of software that can be directly incorporated (hopefully with no or with controlled modification) into a number of different applications. In the research on systematic reuse, this general notion of a component has been broadened in a number of important respects. These refinements are reflected in the replacement of the somewhat overloaded term "component" by the term *asset*. (In general, when we use the term, the more specific "software engineering life cycle asset" is implied.) As the term will be used here, an asset has the following characteristics:

- Assets need not be only source code. Traditionally, the focus of reuse efforts has been on *code*. But in principle reusable software assets can be any workproducts from across the software engineering life cycle, including requirements documents, designs, test plans and data, etc. In fact, most researchers agree that in a number of application areas, use of more upstream materials will yield more dramatic results for reuse.

  Not only work *products* from any phase of the life cycle, but also work *processes* can be reused. A simple example would be a command language script that a developer writes to automate a frequently executed compile/debug/edit cycle of development activity.

- Assets are designed for reuse. Use of ad hoc components, developed for a single context of use, in new applications introduces unpredictability and fragility that often outweigh productivity gains. Such ad hoc or opportunistic reuse is a natural part of the development process, especially in small-scale groups where informal knowledge provides the necessary infrastructure. However, in order for developers to reap the expected benefits of a reuse program and infrastructure, systematic reuse efforts assume that software needs to be specifically designed for a multi-use context, via a process separate from its creation or adaptation for any single context of application.

- Assets can be reengineered from legacy. Note that this does not mean assets must be built from scratch. Software built for existing applications, i.e., legacy software, can be a valuable resource. However, such legacy software must generally be re-engineered for reuse. If multiple examples can be studied, developed for different contexts, then the re-engineering effort has even more material to draw from. In this sense, design for reuse (within an application area with mature legacy systems to draw from) can be viewed as a kind of "parallel re-engineering."

- Assets are most effective within structured collections. Reuse works best when collections of assets are developed, focused in particular domains. It is difficult to get isolated components reused, because the overhead needed to locate, retrieve, and adapt these components for new applications may outweigh the savings derived from their use. If a number of related components are developed for focused classes of applications, however, developers will have stronger incentives to reuse, as they can count on a significant percentage of their application being addressed by the collection of components as a whole. The ODM life cycle assumes that multiple assets will be used in concert in particular applications. We refer to a designed collection of assets targeted to a specific domain as an *asset base*.

- Asset bases present unique architectural problems. Engineering an asset base is a very different problem than conventional systems engineering. When multiple assets have to be used in

a single application, there is generally a non-trivial configuration problem involved. Design of individual assets may be influenced by their anticipated placement within the asset base; thus the asset base itself must be architected and designed in a coordinated way.

Thus most approaches to reuse have come to recognize the critical importance of *architecture* in asset base engineering. In some domains (e.g., conventional math subroutine libraries) the architectural element may not be paramount. In other domains, the architecture may in fact be the primary reusable asset to be created, where components integrated by the architecture must be created by each application developer. In the highest-payoff domains, there is usually an attempt to create both a collection of component-level assets and a set of architectural patterns into which they can be configured. These can include a single, fairly rigid "generic architecture" or a more flexible set of architectural variants.

* The process of asset utilization and adaptation must also be designed. In order to reap the benefits of a managed asset base, it is important that *asset utilizers* not arbitrarily modify the assets in the course of integrating them into new applications. If this happens, then the asset base eventually presents a configuration management nightmare and most of the benefits are from one-time rather than sustained patterns of multi-use.

There are many ways of controlling the utilization process. Use of the asset "as-is" (sometimes referred to as "black-box reuse") is the clearest solution. But a variety of other approaches are possible, including templates or generative capabilities built into the asset base retrieval infrastructure itself.

Domain Engineering State of Practice

Since the seminal work of Neighbors and other researchers in the early 1980's, domain engineering has emerged as a distinct area of research and practice within the software reuse research community [McNi88a, Neig83a]. There are now a number of published approaches to domain engineering, including a range of informal and formal methods and processes [DISA93a, Goma90a, Kang90a, Prie91a,SPC93b]. There is growing agreement that domain engineering is a key component of a more systematic approach to reuse within software organizations.

More recently researchers and practitioners have attempted systematic comparisons of domain engineering methods and technologies [Prie91b, Wart92a, Arango[1]]. Rather than searching for a single best-practice method, such comparative work generally aims to assist practitioners in selecting method(s) appropriate for their organizations and domains. To date, however, no clear consensus has emerged on key differentiators for domain engineering methods. This lack of consensus reflects underlying disagreements about definitions of domain engineering.

Terminology. This lack of consensus extends even to usage of basic terminology. The terms "domain analysis," "domain engineering," and "domain modeling" are used in inconsistent ways in the reuse literature. Usually, domain engineering is the term that refers to the overall process of creating domain-specific assets, of which domain analysis covers the front end, followed by subsequent processes (domain architecture, domain implementation). Other researchers use domain engineering to refer to the back end process only. Still others use domain analysis to refer to the acquisition of domain data and domain modeling to refer to the representation of that data.

ODM uses a few terms with specific meanings, and leaves a few terms out. We call the overall process *domain engineering*, which consists of three main phases, referred to as *domain plan-*

[1.] It should be noted that none of the cited comparisons included any recent version of ODM in the set of methods considered. An up-to-date survey and comparison of methods is needed.

ning, *domain modeling*, and *asset base engineering*. We also use the term "domain modeling" to refer to both the acquisition and the representation of domain knowledge, and sometimes to denote the overall methodological approach or "cognitive style" embodied in the ODM method. We avoid use of the term "domain analysis" because of its imprecision and overloading, and its tendency to suggest misleading parallels with systems analysis techniques. (However, given any of the variant meanings of the term as used within the reuse field, ODM definitely encompasses domain analysis in its scope and hence in many external references is labelled a "domain analysis method.")

Domain engineering is only one part of the context of a broader *reuse program*. (This context is described in more detail in Section 4.0 "Domain Engineering Life Cycle".) Domain engineering, as scoped in this guidebook, does not include general reuse adoption and technology transfer efforts within an organization (which, among other things, increase the organization's readiness to try domain engineering). Nor does domain engineering include ongoing management and evolution of the asset base created, or the changed application development processes needed to utilize the assets. These complementary processes are all needed in order to fully reap the benefits of a domain engineering effort.

Limitations of Current Methods. The systematic reuse community has established ambitious goals for what a domain engineering process should address, such as the ability to support design for reuse for all products and processes of the software life cycle, not just source code. But to date most methods for domain analysis have fallen short of these goals. They are limited by their assumptions of the types of organizational contexts in which domain engineering will be performed and the types of domains that can be addressed. In addition, because these methods were primarily developed by software engineers they embed assumptions about the kinds of software engineering methods and representations to be used in domain modeling, and have also failed to isolate the unique methodological challenges in domain engineering that are *not* adequately addressed by simple extensions to software engineering methods.

3.3 The Essential ODM Approach

The subsections above have provided a brief introduction to the nature of the domain engineering problem and its inherent challenges. We have also explored some inadequacies, both of conventional software engineering techniques and current state-of-practice reuse approaches, with respect to the particular challenges of domain engineering. In this subsection we motivate the overall ODM approach to domain engineering.

Grounded vs. Universal Abstractions

The heart of the ODM method lies in encouraging software engineers to make a certain conceptual "shift of paradigm" to a domain engineering perspective. This shift involves the realization that in designing for reuse, we are implicitly or explicitly designing for some intended *scope of applicability*. The *ad hoc* approaches previously described all reflect a common belief that there is a "right" abstraction for a given component that optimizes for reuse in some absolute or general sense. (And we, naturally, are the ones smart enough to find that abstraction!) In trying to develop reusable software that gets at the right abstraction "in one fell swoop," as it were, we are caught in an illusion that flies in the face of certain basic insights central to ODM:

1) There are a potentially unlimited number of "axes of variability" for any function.

2) Any representation, or implementation, makes tradeoff decisions that fix some of these axes of variability while leaving others variable to some degree.

3) When trying to design software for unanticipated new contexts of use, we have no way of knowing which parameters, which axes of variability, are the right ones for which to design and optimize. This question—which axes are the "right" ones to fix or leave variable (and how variable is variable?)—can only be answered (or "grounded") with respect to some specific assumed context of use.

4) Every time we consider a software component and the *variability* we require, we do so from an assumed contextual standpoint. We imagine the kinds of applications in which we will use the component. The pragmatics of our environment, our constraints and objectives, as well as the limitations of our imagination create boundaries around that assumed context, and those boundaries limit the variability we are capable of conceiving at that time.

   This is as true when we claim we are developing "reusable" software as when we are building custom applications; only the context gets more "invisible" to us. In practice, the developers have a vague notion of the intended scope of applicability for the "reusable" software. But, unlike a single application task where the software produced can be evaluated against a specific customer's requirements, future reusers have no dependable gauge of the perceived requirements to which the software developer was responding other than the comforting assertion that the software is "reusable and context-independent."

5) We do not escape from this contextual grounding by eliminating levels of detail from our description. In other words:

   Abstract is not universal.

6) We can become aware of some aspects of that context, but can never jump out of it entirely. Accepting this requires both an acknowledgment of our limited vision and the fact that we are always grounded in a contextual framework. We rarely try to characterize the contextual assumptions behind our approach while we are caught in the trap of trying to "universalize."

So much for the "bad news"! These principles might appear to lead to a certain pessimism about attempting to develop reusable software in any meaningful way. There are an unbounded number of "right" abstractions; we can't design for all of them; and hidden contextual assumptions will affect the way we design. Then how do we design a process that helps reveal and document our hidden contextual assumptions? Simply exhorting ourselves to "document our assumptions" probably won't be very effective; after all, our contextual assumptions, by definition, are the things "invisible" to us at a given time; mere good will won't render us suddenly omniscient.

However, we can make progress by recognizing some additional principles:

1) Every time we develop, adapt, use, or study a software component, new knowledge is created in these acts themselves. This new knowledge inevitably changes the contextual field from which we are working; and this can reveal new possibilities, new axes of variability.

2) Making a conscious effort to reflect on the contextual assumptions we make when we develop software, and document that contextual information, can immediately improve the quality of the software and its ability to be reused more predictably in new applications. The reusability of a work product can only be assessed relative to explicit documentation of its intended scope of applicability.

3) Defining an explicit *multiple-system* intended scope of applicability for a software work product is essential to creating a reusable asset in a systematic way. The process is different from designing for a single, assumed context of application; it is also very different from designing for reuse in some absolute sense. It means building to a specified and constrained range of variability.

4) Studying a set of related workproducts developed for different, specific application contexts is a very effective way of facilitating the reflection and discovery process that reveals previously hidden contextual assumptions. By looking at a carefully chosen set of example data, and paying attention to both the commonalities and variabilities across this set, it is possible to use the discrepancies and deltas as a "cognitive pattern detector" to raise the implicit assumptions to awareness.

The ODM approach is therefore designed to lead to discovery of "grounded" rather than "universal" abstractions. If the first set of principles acknowledges our limitations as designers, these ideas celebrate our capacity to learn and discover. The overall process can be viewed as an inductive and empirical way of arriving at abstractions that will, in the end, have that ineffable quality of "rightness," naturalness and intuitiveness that we otherwise try to arrive at strictly through our talent as software engineers.

It is this goal that makes domain engineering using a set of explicit exemplars a unique process and not simply a form of "variability engineering," or simultaneous system engineering to multiple requirements. Such a task probably could be performed by "scaling up" existing techniques to handle a more complex set of requirements. In domain engineering, however, the multiple data points are selected in order to help arrive at a domain model that will be usable beyond the examples studied. In fact, the ODM method embodies a working hypothesis that might be considered the complement to the "reuse paradox" that began the discussion in this section. The "Happy Paradox" is the following:

> If you design for multiple contexts of use rather than a single point of use, and document these multiple contexts, you are likely to get a component that is more reusable, even beyond the multiple intended contexts of applicability. The cognitive processes of comparison, discovery of commonality and variability, reveal contextual assumptions that would otherwise stay invisible.

Of course, you never unearth all contextual assumptions through this process. But every assumption you do discover will directly improve the software's design and reliability. Either you will design in variability for the newly discovered context of use, or you will fix that assumption and document it (because it is now visible to you). Users of the component then will have a better chance of being able to reuse the component in new contexts, or at least to find out in advance that the component is unsuitable and why, rather than after a lot of adaptation effort.

3.3.1 The ODM Concept of Domain

We have seen that the notion of an "explicit, intended multi-system scope of applicability" is an essential part of a systematic approach to design for reuse. At its most essential, this indicates the functional role of the concept of a domain in the ODM approach: i.e., the "domain" for a reusable component is its intended scope of applicability.

This somewhat Spartan definition conflicts in important but subtle ways with generally accepted notions of "domain" in various technical communities, including but not exclusively the reuse community. Unfortunately, this generally accepted notion masks a good deal of confusion over the term. In fact, the question of what is meant by the term "domain" continues to plague the reuse field. The discipline (such as it is) is a hybrid of different techniques drawn from widely different fields both within and outside of traditional software engineering concerns. The term "domain" is overloaded, connoting different things to members of these different communities. We need to provide some background in order to clarify the ways in which the ODM concept of "domain" concurs with and diverges from conventional understandings of the terms.

A Thumbnail Sketch of the Concept of "Domain"

For the purposes of this discussion, we'll compare the notion of "domain" used in several of these broad communities from the fields of linguistic, cultural and/or anthropological research, Artificial Intelligence (AI) and object-oriented (OO) technology, and software reuse. We will discuss two broad concepts of "domain-ness" illustrated schematically in Exhibit 2.

Domain as the Real World

When many researchers use the word "domain" they mean, in some sense, the "real world" where real work gets done. Software systems get developed and deployed in these real world settings. Software developers have "system knowledge" about programming languages, design methods, implementation details, architectures, etc. Practitioners, end-users of the system have "domain knowledge" about medical procedures, helicopter engines, and all the other messy details of physical reality. Some of that domain knowledge is weakly reflected in codified form in software systems, but much more of it stays embedded in expert knowledge (or perhaps in textbooks and other documented forms).

The knowledge engineering and AI community was one important source for this understanding of the term "domain." Typical knowledge engineers were investigating settings where people performed with "expert" competency. When the word "domain" was used in this setting, it generally referred to this area of expert competence in the non-automated world of technical practice that was the focus of the research.
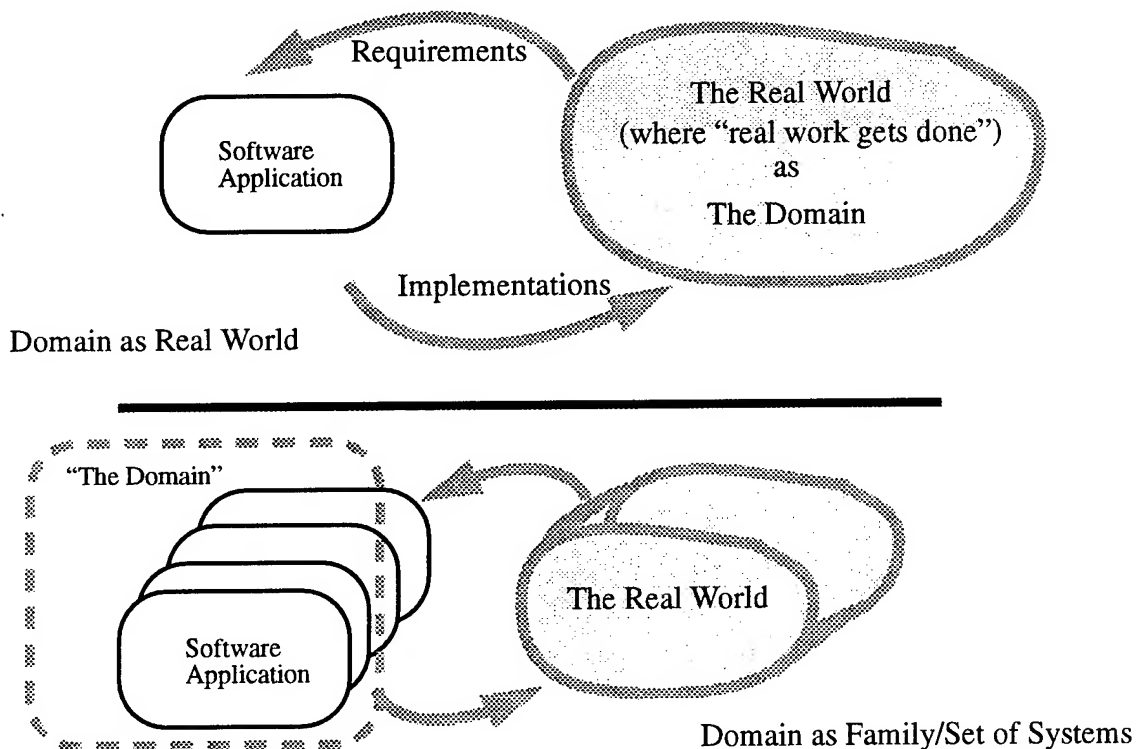
Exhibit 2. Two Different Concepts of "Domain-ness"

The tasks the experts performed were generally ones that had *not* been previously automated with conventional software systems. (This made them attractive applications to demonstrate the power of AI techniques.) Historically, the mainstream of expert systems (ES) work was not applied to studying the expertise of software developers. (This was a related niche sometimes termed "knowledge-based software development.") Hence, within the ES community, the word "domain" usually connotes the real world realm of practice as opposed to the system engineering world. For example, in the medical application area, the "domain" would include the world of doctors, patients, hospital rooms, blood pressure, etc.

To some extent, this basic theory of "domain as the real world" is also used in other technical communities, such as certain software methodologies. Also, at least on the surface, the object-oriented (OO) community's use of the term "domain" closely resembles the ES community's usage. As OO programming techniques pushed into OO design, and then into OO analysis, it became increasingly important for OO practitioners to describe real-world scenarios and system requirements in ways that were in keeping with their preferred approach to system-building. An OO-oriented practitioner, then, will often interpret "domain analysis" as an object-oriented description of the entities and transactions in the "real world," a description that forms the basis for implementing an OO system to support those real-world activities. One indication that this theory of "what a domain is" is being applied is a statement like, "Domain experts don't know or care about software."

Domain as Set of Systems

When the term "domain analysis" began to be used in the software reuse field, it borrowed some aspects of usage from the AI community. But the context of use was very different. Generally, reuse-oriented domain analysts were interested in creating models to facilitate the design of software to be reused across a class of systems. Unlike the AI setting where the "real world" was the subject of the domain, in reuse-based domain analysis *software systems are themselves the objects of study.* A domain is usually informally defined as "a family or set of systems including common functionality in a specified area." So, for example, where an AI researcher might expect a domain description to include "actors," "agents," and other elements of real-world scenarios of practice, a software-oriented domain analysis might include "objects" and "operations" that reflect components of the software systems themselves. For example, a domain model of window-management systems would probably include terms for "windows," "pull-down menus," "open window," etc. The domain model might not address the computer user interacting with the window management system, or workers interacting with each other. In the medical area, in the "real world" theory the domain experts would be the "practitioners," the doctors, nurses, paramedics, etc. In the "set of systems" theory, the domain experts would be system engineers who had built one or more of these kinds of systems. Modeling commonality and variability across systems in the domain is usually the focus, to facilitate engineering of components with sufficient adaptability to be reused in multiple applications.

The ODM Perspective

In ODM, domain modeling is a social and conceptual process that can be applied to any "realm of discourse" where commonality and variability of multiple exemplars is examined.

In the most general sense, a domain in the ODM context can be any field of inquiry or realm of discourse that has a common, defining categorical definition and a set of elements that are evaluated to be members of the domain by virtue of sharing those defining features. The overall, high-level ODM process is broad enough to address this extremely general definition for domain.

This guidebook, however, elaborates the ODM method at a level of detail and including a set of workproducts that assumes the domain of focus is a domain in the software reuse community's sense of the word, i.e., domains focused on software system functionality, as shown in Exhibit 2. We assume a set and/or class of systems that exhibit some common functionality, either across the entire scope of concern for the systems or within some part of the systems only. The domain is a formal construct that encompasses this functionality. Note that this definition does not rule out domains in either the "real world" sense or the "set of systems" sense; but neither does it assume one or the other interpretation.
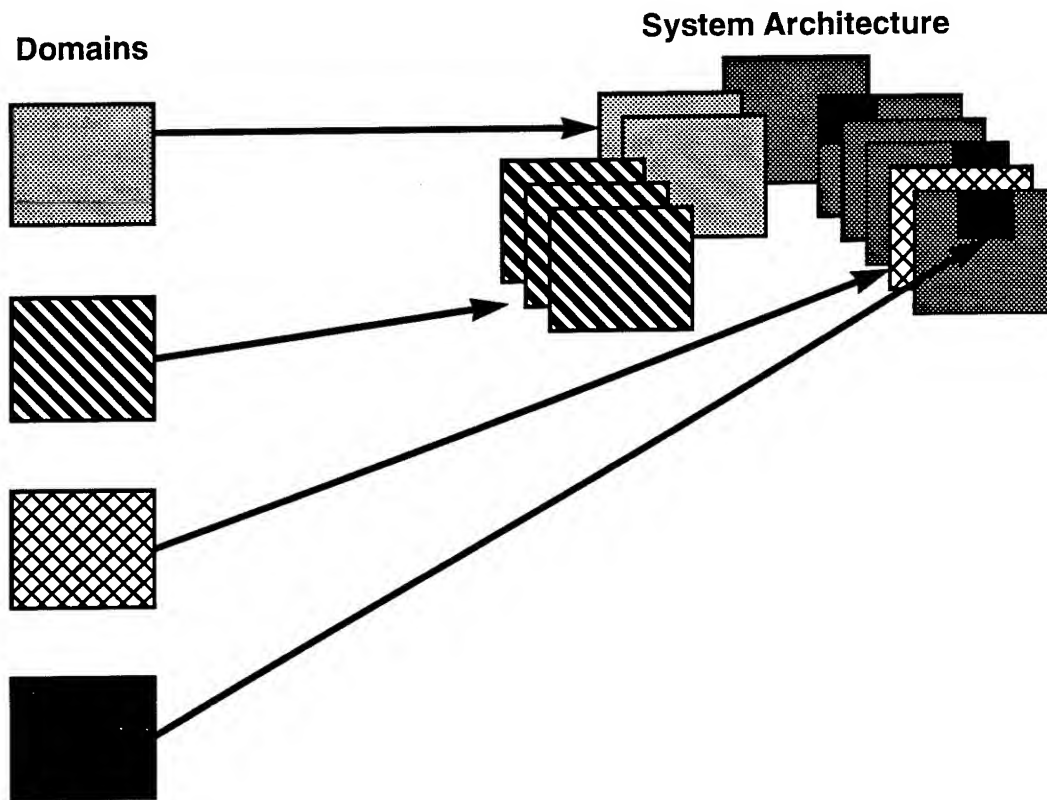
Exhibit 3. Large Systems Span Many Domains

Ethnographic Domains

The ODM definition of domain may appear somewhat "relativistic." In fact, it draws inspiration from an ***ethnographic*** approach derived from cultural studies (sociology, anthropology, linguistics). Researchers working out of an ethnographic approach are generally interested in eliciting a community's own definitions and ways of making meaning of their culture, rather than collecting data to confirm or refute pre-determined hypotheses about culture (as in experimental science or quantitative survey approaches). As part of ethnographic data gathering, researchers study the language, or practices, or social structure, of a given cultural group. Though the scope of this inquiry can vary widely, we can consider it as a "cultural scene." Within that scene, people interact and use language in ways unique or at least particular to the scene. Researchers remain on the look-out for broad categories of experience that are of interest to people in the community being studied. Some evidence that a category is of interest is the presence of a number of terms that differentiate concepts within the broad category, where the "differences make a difference." These categories are what such cultural researchers are likely to term "domains."

For example, the ethnographer James Spradley researched American tramps and discovered a rich domain called "flops" [Spra79a]. The paraphrase "places to sleep" does not capture what the term "flop" means for a tramp, since many places to sleep might not be flops, etc. Through studying the domain of "flops" Spradley got to understand something of the way that people in that cultural scene understood their environment and their relationships with other people.

Once domains are identified, and terms are collected and included within the domain, the domain is structured according to one or more kinds of semantic relationships. One of the most prevalent of such relationships would be the class-sub-class, or specialization relationship (sometimes called the "is-a" relationship). When a cultural domain is structured in terms of this relationship, a taxonomy is produced. But not all domains are best modeled as taxonomies. In principle one could have a domain with just two or three distinct terms, but it would appear to be not a particularly useful, insightful or interesting way of partitioning domains.

In this research context, a domain can best be described as a "coherent realm of discourse." People in the community need to recognize the domain as being of interest, a distinct topic of focus, and one with significant depth of detail or complexity. Knowledge of the meanings of terms within the domain is part of the knowledge required to be a competent member of the community.

What are the interesting features of a domain in this context?

- A "covering term" or domain name drawn from the language of the community;

- Recognition by the community of the domain as a distinct category or focus of discourse;

- Existence of a set of "included terms" within the domain, also drawn from the language of the community, with enough included terms to suggest that the domain is the focus of some interest, attention, and differentiated competence.

- At least one semantic relationship that structures the terms and places them in meaningful relation to each other and to the covering term.

The domain terminology is generally collected from or cross-checked with several individual *informants*, so that it represents language used in common within the community and not just an individual's personal way of structuring information for themselves. That is, it reflects cultural and not merely personal knowledge.

General ODM Definition of Domain

While focusing on domains in the software-specific sense, ODM incorporates much of the ethnographer's sense of a domain as an expression of cultural meaning-making. In this case, the culture is that of the software engineers who build particular kinds of applications and the practitioners who use the systems in their workplace settings. The ODM process attends to many of the same methodological details required of good ethnography, such as adjusting for possible observer bias, and paying attention to the specific terminology used within the community. ODM applies this ethnographic approach to modeling the domains that software engineers are interested in. This includes, potentially, studying the "cultural scene" of the end-users of the system (the "real world" domain), the cultural *artifacts* of the system builders (system requirements, designs, implementations, and other documents) and, if possible, the developers' cultural scene as well (the setting where requirements are analyzed, applications are built and maintained, etc.)

Organization Domains

Domain engineering, of course, takes place for the most part in business environments where companies have strategic objectives to be realized. ODM addresses this issue with the recognition and assertion that domains are always socially situated or socially constructed; that is, they are always shaped by the context of multiple overlapping communities of use, development, maintenance, customization, and application. As implied in the name of the method, ODM deals not with "domains" in the abstract but with *organization domains* as depicted in Exhibit 2. ODM uses explicit modeling of social and organizational context to ground the entire modeling life cycle, from articulations of domain engineering objectives and domain selection criteria, domain identification, definition, and scoping, through domain modeling and up to and including selection of technologies for asset implementation. Organizational context also establishes entry and exit criteria for modeling activities otherwise quite difficult to bound.
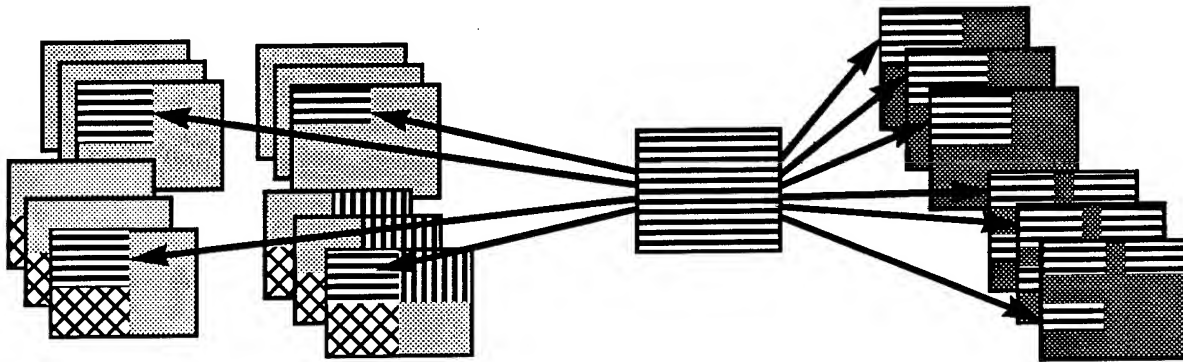
**Domain provide new bridges for interconnecting
organizational entities with technical systems they
create and utilize —
not via control or process flow, but _KNOWLEDGE_**

Exhibit 4. Organizations, Domains, and Systems

The term *organization* is used in this guidebook as a place holder for various entities of the business context in which domain engineering is performed. This could be a single organization such as a software product company or contractor organization, a specific department or division within a larger organization, a group of organizations linked as suppliers, value-added resellers and customers, a market segment, a research center or university, a consortium, or even a community with shared interest in some specific technical area such as a standards body.

The term *system* is used in this guidebook interchangeably with *application*, to denote an application designed for a single context of use. There is no intent to imply a *systems* (i.e., hardware, software, practitioners) versus *software*-only distinction.[2] What constitutes a single application will, of course, vary from organization to organization. However, an organization domain usually represents functionality that spans multiple applications as defined *in the organization's own terms*. In the ODM perspective, domains are designed and negotiated strategies for linking together organizations and systems in ways that can be independent of structural lines of division in both organizations and systems. An organization domain embodies a "socio-technical" linkage

[2] The extent to which domain engineering techniques are software-specific is a complex issue beyond the scope of the guidebook.

between the organization contexts that create, manage and use software systems and the systems themselves.

What kinds of domains do we want to define?

The goal of domain engineering could broadly be defined as optimizing the software development process over a "window" of multiple applications within an organizational context. Of course, there are myriad ways of improving software development practice: general process improvements, introduction of standard methods and software engineering environments, use of more advanced languages. Domain engineering focuses on the "application-specific" content of a targeted set of systems, and aims to create reusable software resources that, in one way or another, eliminate redundant development work. The optimization could take many forms, but generally involves creation of some reusable components or assets that can be incorporated into multiple applications.

Since the goal of reuse is some form of optimization of the software development process, ***domain engineering initiatives*** fall in a sometimes uncomfortable no-man's land between "production" (in the system of task-oriented development work whether in developing custom applications or products) and process improvement initiatives. Although its place within the overall reuse program context is quite specific, domain engineering nevertheless can have a scope and impact much broader than that traditionally associated with an effort viewed as a "technology initiative." This means that it can be difficult to know how to launch domain engineering projects, where they should be housed in the organizational structure, how they should be funded and evaluated for return on investment, etc.

Every domain is defined and modeled in the context of some stakeholder interests (organization domain) and making this context explicit helps guide the process more smoothly. The most interesting domains draw new boundaries within and across both software applications and organizations. This can involve significant intervention in the business processes and even the structure and strategic goals of the organization. In particular, there may be organizational changes needed to introduce an asset base into a development process where one previously did not exist. Domain engineering may also involve significant "culture change" on the part of the organization and individual engineers. But this can also be a tremendous value added for domain engineering, which can be viewed in this sense as a specialized form of business process reengineering, applied to the processes of developing particular families of software applications. In at least some cases, a properly scoped domain engineering effort can be an excellent catalyst to spark broader acceptance of systematic reuse within the organization.

Subsystem level domains. Many DA approaches assume that the domain encompasses a set of entire applications, a so-called "family of systems" approach. ODM explicitly allows for, and in fact, encourages, identification and selection of domains at the subsystem level. it includes specific processes for addressing the extra scoping steps that must be performed to support this finer-grained notion of domains.

ODM does not rule out selection of an application-wide domain, but does facilitate careful consideration of the risks involved in this strategy. In most cases, even if this is the eventual goal, an incremental approach will turn out to be the best strategy. Our experience is that domains defined at the level of granularity of an application or system for the organization are almost always so big that they require an untenable level of commitment on the part of the organization.

There are other benefits to this finer-grained approach. If domain modeling is done as a precursor to engineering assets for reuse, it is at this subsystem level that it will often make most sense to do reengineering. Assets implementing a particular variant of the subsystem functionality desired

can be incorporated into the reengineering efforts for legacy systems, where an approach based on entire application architectures may create obstacles for such incremental evolution of legacy systems. At the same time, subsystem level assets can be engineered to be useful outside of one "family of systems."

3.4 Key Aspects of the ODM Method

So far we have motivated the need for a distinct discipline of domain engineering, presented the key concept of a grounded abstraction as an explicit, multi-system intended scope of applicability, and provided some background context for the ODM notion of an organization domain, which rests on this key concept. Before we can further explore the advantages of ODM in the context, we must consider some additional questions or challenges raised by the concept of an organization domain.

Challenges Presented by Organization Domains

The emphasis on a multiple-systems scope as described above raises a methodological challenge in domain engineering: that of domain boundary definition and scoping. In fact, these issues apply in any engineering context, where designers' scoping decisions are usually implicit, informal and private. Being more explicit, systematic and public about how scoping decisions are made would benefit any software engineering project. However, systematic scoping is critical to the very feasibility of domain engineering.

Problems of selecting scope are more tractable when the development effort is guided by a clearly defined problem space (e.g., user/customer/market requirements) and/or a clearly defined technology or solution space (e.g., architecture, technology availability, development philosophy). In a problem-driven engineering context engineers seek solutions with respect to a somewhat stable problem. The single intended context of use establishes a scope within which requirements can be defined. In a product- or technology-driven context there is a stable, if evolving, solution space for which engineers seek problems, or rather opportunities, for application.

In domain engineering there is no single system context to establish clear entry and exit criteria for modeling tasks. Where does the scope in a multiple-system effort come from? How many systems are enough, or too much? These issues are compounded when the potential scope can include both legacy and proposed new systems. In fact, typically in domain engineering contexts both problem and solution spaces are *simultaneously dynamic*. The nature of this methodological dilemma is illustrated in Exhibit 5.

There are many simple examples outside of the software engineering context for these kinds of situations, e.g., simultaneous arguments over who will be a member of an organization and the proper mission of the organization. Certain breakdowns are symptomatic of these situations, such as a tendency for boundaries to "wander", or for circular conflicts that alternate between discussions about definitions and decisions, etc. In domain engineering, these process breakdowns can have major repercussions at the concrete level of system architectural decisions.

Independence from architectural and design decisions of specific legacy systems is critical to addressing domain engineering risks, as these decisions may not scale up to be applicable across the entire range of systems desired for the domain. On the other hand, in mature domains obtaining maximum value from legacy systems is essential. Otherwise domain engineering amounts to system design with variability, dependent on the architecture and design expertise of asset base engineers who may not know the rationale behind the design of existing systems. Since domain assets, if accepted, will be applied across many systems the potential impact of poor design is
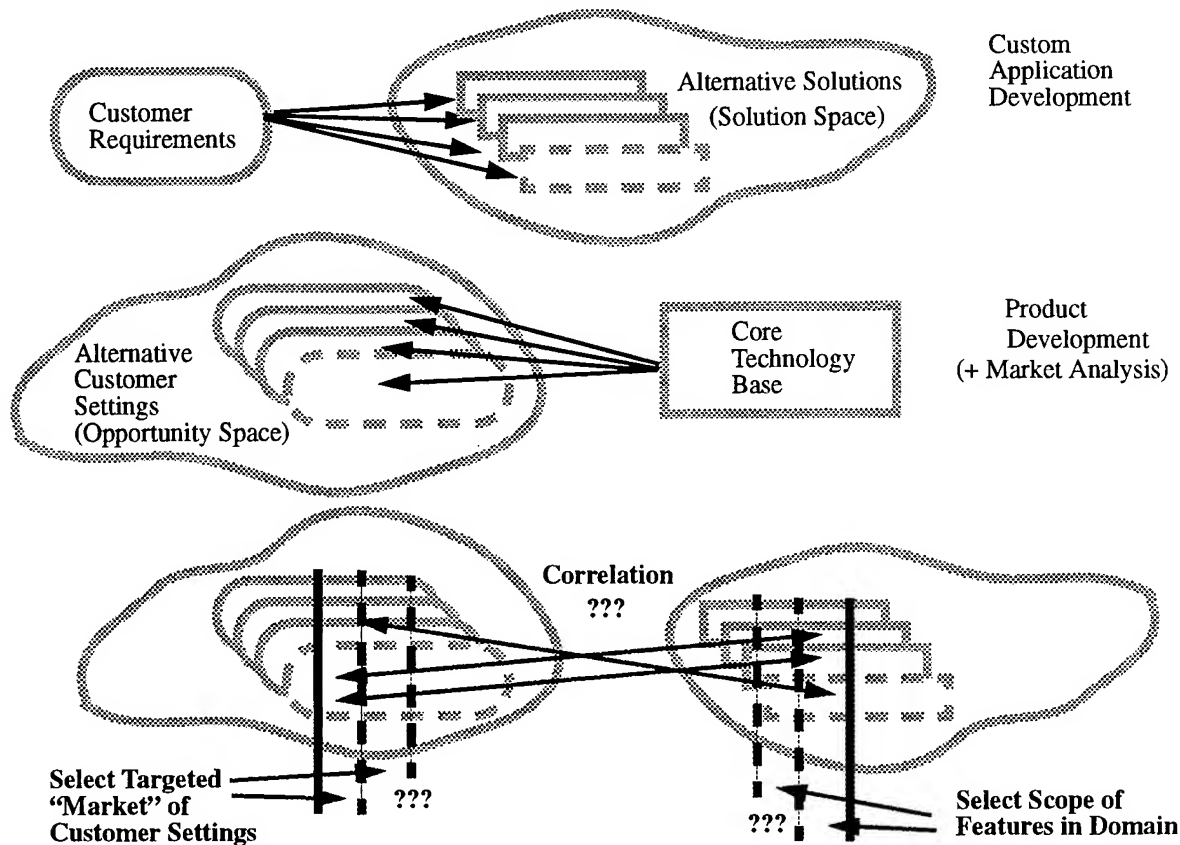
Exhibit 5. Application, Product, and Domain Engineering

much broader than if made within a single system development effort. How can we derive value from legacy information that is available without being unduly influenced by this legacy in cases where it is not appropriate to carry old architectural approaches or implementations forward to new systems?

Distinctive Features of ODM to Answer These Challenges

We can now explore some of the specific features of the ODM method that address the challenges inherent in the domain engineering task as scoped above. These features include:

- Iterative scoping;

- Stakeholder focus;

- Exemplar-based modeling;

- Explicit modeling of variability across exemplars;

- Emphasis on *descriptive modeling*;

- Methods for context recovery;

- "Settings" as context;

27

- Feature binding sites; and

- Innovation and the "archetypal" domain.

These are each discussed in more detail below.

Iterative Scoping

ODM's approach to the problem of systematic scoping involves structuring the entire domain engineering life cycle as a series of incremental scoping steps. Each step builds on and validates previous scoping decisions and offers different opportunities for refinement and focusing. Throughout the life cycle, scoping decisions serve as a risk mitigation strategy, constraining and managing the process. At the same time, design and modeling decisions made at each point have a rich foundation of data on which to draw because of previous steps. This view of the ODM life cycle is illustrated in Exhibit 6.
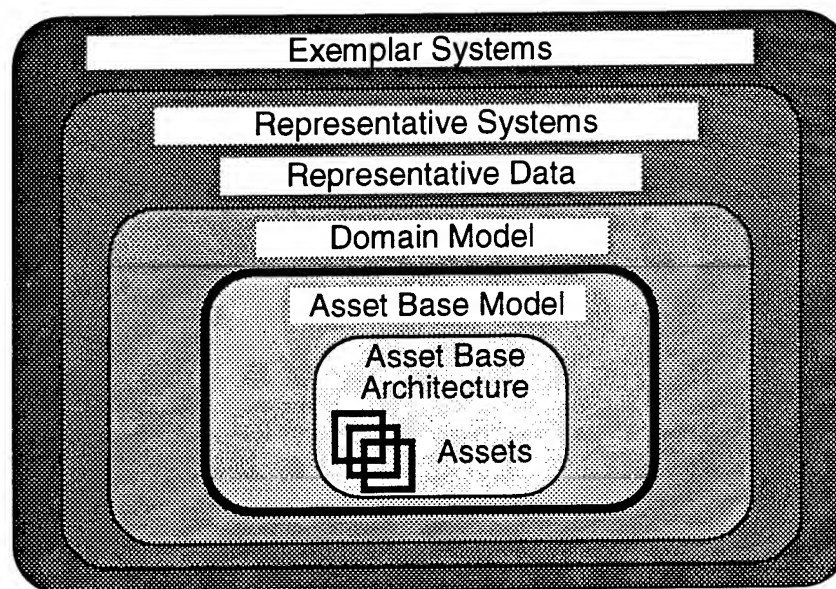
Exhibit 6. Iterative Scoping Steps in ODM

After selection of a domain of focus for the project, the definition process involves documenting a specific set of exemplar systems that help to bound the domain.

Defining and scoping the domain is distinct from scoping the data that will be acquired to model the domain. Once the definition is stable and data acquisition begins, a further scoping step takes place in selecting a subset of exemplar systems as a *representative systems* set for intensive analysis. Further scoping activities are performed to select the best cross-section sampling of data to gather from various system life cycle phases of these representative systems, and which specific individual models will be developed from the data.

Since the domain will typically be only one constrained view of the overall system functionality, scoping activities continue throughout the modeling process. The resulting set of *descriptive models* is formally integrated and interpreted, establishing rationale and contextual information about variability within the domain. The final domain model may only formalize conceptual material within certain selected areas of interest in the overall domain scope.

With the cross-over to asset base engineering, additional scoping takes place. An asset base may involve support for only a subset of the features described in the domain model; this subset is captured in the asset base model derived from the domain model. The implementation and technology decisions made for the asset base further constrain the *feature space* of the asset base model, as expressed by the asset base architecture. After this architecture has been specified, asset implementors may still choose to develop only a subset of assets specified in the architecture.

Successive levels of scoping in ODM should result in a narrowing of the domain engineering problem into specifications that can be implemented using the appropriate supporting methods of system engineering, augmented with variability engineering techniques.

Stakeholder Focus

ODM grounds all aspects of the domain engineering life cycle in an up-front analysis of the organizational stakeholders and objectives for the effort. Domain planning in ODM begins by modeling the set of *stakeholders* for the domain engineering project: specific groups and/or individuals with interests and objectives relative to the project and the domain, as well as diverse viewpoints, experience and terminology. Documentation of stakeholder context grounds domain selection by formally modeling the strategic business setting in which the domain engineering project is being performed.

There are multiple stakeholders in any system development effort. But regardless of whether the stakeholder context is entirely internal to a single organization or involves interactions between several organizations, domain engineering almost always involves the dynamics of multiple stakeholder interests and objectives. This makes strategic alignment an ongoing challenge in the process, inherently more complex than simply "keeping the customer needs first" in a single system development effort, or focusing on the needs of "the organization" in an organization change management effort. Domain knowledge is typically distributed, not only across multiple systems, but across organizational, departmental, project and product line boundaries. This creates complex trade-offs and boundary negotiations between diverse stakeholder interests in domain selection and later modeling decisions, where requirements may differ in priority or even be inconsistent, and even basic domain terminology may be interpreted in diverse ways. This is reflected in ODM's re-consideration of strategic interests of stakeholders at critical points throughout the domain engineering life cycle.

Exemplar-Based Modeling

Rather than trying to generalize from a single example, or speculating in the abstract about a "general" solution, ODM works from an explicit set of examples of the domain, or *exemplars*. This corresponds closely to the way that people form concepts on an intuitive basis, but makes these intuitions public, hence shareable and verifiable.

We use the term *exemplars* in contrast to *assets* to denote a workproduct created for a single application context. The overall ODM process can be viewed as a transformation or restructuring of exemplars into assets, targeted to a specific domain.

The mapping takes place through the intermediary definition of domain *features*, significant differentiating capabilities across systems within the domain. Wherever possible, complex features are deferred to *related domains*, to keep a tight focus on the models and the domain scope. Implementors of reusable assets, specified in terms of domain features, can trace backward through descriptive models to find candidate *artifacts* for reengineering based on this traceability. Thus assets are designed for reuse in the context of the asset base as a whole; but the asset implementation process can extract considerable value from legacy systems.

Advantages gained from multiple exemplars don't increase linearly. The good news is that the greatest learning and contextual discovery seems to take place with the first few exemplars studied. With a well-chosen set of examples and some techniques for focusing attention on the variations, we believe that the lion's share of the value of the exercise comes with the first few examples studied. In any learning task, you may not need to complete all the work you plan up front in order to "get it." Once you "get it" you are done (with some methodological reservations). A metaphor that may help here is that of triangulation. The more data points you have, the more you can fix a given point's location. Each exemplar system you study has the advantage of providing a "fix" from a different contextual standpoint. After doing a comparative domain modeling effort on a dozen applications, studying a dozen more (unless chosen via specific "diversifying criteria) is not likely to yield the same number of discoveries. This holds out the hope that the process could be describable, boundable, and formalizable.

Domains are comparative over multiple exemplars. Perceived similarities and differences among multiple, similar entities (individuals) within the domain are a primary focus of ODM. If one were to study a particular hospital setting, modeling doctors, nurses, workflow and scenarios in the sense of an individual situation, this would not in and of itself constitute an ODM-style "domain." Only if one studied *multiple* settings of this kind, or, internal to one setting, studied, for example, the various kinds of triage events a doctor might need to respond to, does one enter the realm of domain modeling in the ODM sense. ODM usage implies that a primary kind of relationship among individual exemplars in a domain is taxonomic, "kind-of" relations.

Exemplars vs. Legacy Systems. ODM's emphasis on working from explicit exemplars can be confused with an emphasis on legacy systems. The confusion is understandable, because the exemplar-based approach dovetails neatly with the many other practical reasons to systematically study legacy systems in a mature domain. Legacy systems will form important exemplars for study, but exemplars need not be legacy systems. However, the insistence on explicit exemplars is really a separate issue determined by methodological issues. The exemplar systems selected in ODM can include *both* legacy systems and requirements for new systems.

Explicit Modeling of Variability

ODM encourages modelers to maximize variability in the descriptive phase of modeling. The suggestion to explicitly model genealogical relations between candidate representative systems, for example, acknowledges that domains can be defined to encompass systems from quite distinct lineages and organization contexts. Similarly, modelers are encouraged to choose representative systems that have diverse structural embodiments of *domain functionality*, e.g., an encapsulated subsystem in one context, distributed functionality in another.

The rationale in this intentional seeking out of high diversity is to use the *descriptive modeling* process to generate as much insight as possible about the potential range of variability. This would be a high-risk approach if there were not complete independence between the descriptive and prescriptive phases. Because asset base engineers can select whatever subset of the DOMAIN MODEL is strategically appropriate, they can scope the asset base in a variety of ways starting from a DOMAIN MODEL that is rich in variability. Separation of design processes into descriptive and prescriptive phases is therefore a recurring strategy at many levels throughout the ODM life cycle. Wherever it appears it is being used in part as a technique to regulate and manage variability.

Descriptive Modeling

In order to emphasize the particular "cognitive style" necessary for domain engineering, the ODM process model separates the life cycle into two distinct phases: a *descriptive* domain modeling phase, and a *prescriptive* asset base engineering phase. In *descriptive modeling*, we are selecting

and studying a set of example systems for the domain in order to derive the shape of the domain "space" itself. After having established this, we are then in a position to carve out particular sub-regions of this space for engineering efforts.

Descriptive modeling documents what experts have learned about how to build particular classes of applications through the experience of developing multiple systems. In descriptive modeling, knowledge about the domain is obtained in part by analyzing legacy systems through the intermediary definition of domain *features*, significant differentiating capabilities across domain systems. Modelers document commonality and variability in system structure and function and attempt to recapture the rationale for decisions embedded in those systems.

Descriptive domain modeling is thus somewhat analogous to **reverse engineering** in a reengineering context; or to the "current physical" and "current logical" models (sometimes called the "as is" model) in Yourdon systems analysis techniques. However, there are significant distinctions as well. In ODM descriptive models can contain information about requirements for new and anticipated systems as well as legacy systems. Conversely, customer systems for the asset base might include legacy systems to be reengineered.

The descriptive model is best thought of as a language for expressing a *coherent space of possibilities* within the domain. This model can include aspects of *both* the problem and solution space, as well as interpretive rationale connecting them. The key principle of the descriptive phase is that no binding decisions or commitments are made about the functionality to be supported in the asset base.

Note that the final domain model is extended beyond a purely descriptive model through innovative transformations applied directly to the model. This extension step is necessary to ensure that the prescriptive asset base model is always a subset of the domain model. This step is described more fully at the end of this section.

The asset base engineering part of the life cycle marks a shift to **prescriptive modeling**, where binding decisions and commitments about the scope, architecture and implementation of the asset base are made. The descriptive *feature model* is transformed into a prescriptive set of committed features for implementation. The prescriptive phase begins by re-scoping the range of functionality to be supported by the reusable assets to be developed; commitments are made to a real set of customers for the asset base. These commitments cannot be finalized in the initial project planning step, because they depend on data that emerge from the descriptive modeling phase. This scope is documented in the **asset base model** which therefore represents a subsetting of the final extended domain model.

This prescriptive model leads to the development of an **asset base architecture** to support the prescribed variants. Prescriptive features are mapped onto the structure of the asset base and to sets of specifications for particular assets. By preserving traceability from features back to exemplar artifacts, asset developers can gain access to potential prototypes on which to base development of new assets. In addition, because assets are described within the asset base in terms of the feature model, asset utilizers can eventually retrieve components based on the same **descriptive feature** language.

Why is Descriptive Modeling Hard?

The ODM process is designed, not only to get us to do certain activities in a certain order, but also to *stop us* from doing certain other activities that may be habitual, but are not useful in domain modeling. For software engineers, their very software expertise, if mis-applied, is one such set of habits that may be a liability in domain modeling. There are places in the process where it is critical *not* to think of the task from a software engineering standpoint. This goes against the grain,

and at times the purpose for the task will be unclear unless you continually remind yourself of the domain modeling process goals.

As a hypothetical example, suppose you are studying a domain with a number of legacy systems, and you are looking at the way several systems designed the module hierarchy. As you build the domain model, you think: "This design is really pretty poor. I don't see why they didn't encapsulate these two modules together; this is probably an artifact of the old operating system they were running on." You therefore change the design as you see fit and encode that in the model.

The reason this "breaks" the domain modeling process has absolutely nothing to do with whether your design decision was a good one or not. In fact, because it's a domain and not a system *there's no way to make that judgment call*, and therein lies the problem. You have "changed the data" based on a contextual interpretation that may not be true, or may not be the whole truth. For example, what if there were several other reasons for the legacy design that was changed, that you will now not discover?

This is why ODM insists on a first phase in domain modeling that is descriptive in nature, where design decisions and options are described but not committed to. It is extremely difficult for software engineers to work this way. It does not correspond to the kinds of tasks they have been trained to think of as "productive work" or "making progress." In fact, only by giving engineers a fairly formal process to follow, with explicit guidelines, is it possible to get them to stop designing long enough to do domain modeling!

Context Recovery

A key function of descriptive modeling is *context recovery* to identify and make explicit contextual information embedded within artifacts. Software created for a single application embeds hidden "contextual" dependencies that create problems when developers attempt to transfer that software to new applications. Doing context recovery for system artifacts can render them more dependable and predictable, even for *ad hoc* reuse, by making dependencies explicit. (It does not remove these dependencies; of course. This requires the further step of reengineering artifacts into assets.)

Documentation of context is also important throughout the modeling process. The language, values, assumptions and history of each relevant community introduces contextual information into software artifacts that invisibly constrain those artifacts. This "hidden context" in software artifacts is one of the primary sources of uncertainty in *ad hoc* reuse of legacy components. Because many constraining assumptions come from the cultural context in which software artifacts are developed and used, the domain modeling process must identify and make explicit how this kind of information is embedded within the artifacts.

Unearthing this implicit context, fundamental to overall aims of domain engineering, requires a rich set of data acquisition techniques. Most DA methods recognize the need to gather information from both artifacts and interviews with domain experts. The ODM approach to data gathering reflects the principle that a domain model can only be validated relative to explicit documentation of the information sources from which it was derived.

Several complementary techniques can be used, including: examination of legacy artifacts; structured interviews with *informants* (including not just domain "experts", but system users and other practitioners in the domain); and *ethnographic techniques* such as direct observation or participant observation of task work flow in the domain. Use of complementary data gathering techniques creates robust opportunities for cross-validation. Strategic use of multiple representative

systems is also integral to obtaining the benefit of certain cognitive insights that are elicited through perceived variation across similar systems.

This cognitive effect means that data acquisition must be treated as an active intervention in the stakeholder communities for the domain. Especially in group interviewing settings, informants may interact with people in the organization (or from other organizations) that they would never have met in an ordinary project-centered context. This can result in significant learnings about the domain, for modelers and informants.

Settings

Scoping and descriptive modeling approaches alone are not sufficient to aid context recovery. Embedded assumptions can be found via identifying, describing, and modeling the cultural or contextual aspects of the domain as key descriptors along with software artifacts. The difficult part is getting a handle on just what is meant by this "context." Up until now, we have used the term "context" somewhat loosely in the discussion. The term "context" is perhaps as overloaded and fraught with confusion in the domain engineering field as the word "domain" itself. A major contribution of the ODM method is a refinement of certain core concepts that make the problem of context more tractable.

We use the term *setting* to denote any work environment relevant to the domain of focus.[3] Although the notion of setting could be applied to any domain, it has particular importance for dealing with domain engineering in software-intensive domains.

Systems span organization settings. It is inherent in the nature of software that creation and use of applications take place in different settings. Like other products and information, software systems are created in, exchanged among and used in various organization settings.

For any software application, there are a series of settings that involve the life cycle of that application. This is most typically referred to as the "software/system/application/project life cycle." Different elements in this life cycle are called "phases." We prefer the term "setting" for individual phases, because the relations between settings need not be in temporal sequences. For example, an application developer might have linkages to many different customer sites. For convenience we retain the term *system life cycle* for the overall configuration of settings.

Exhibit 7 depicts a minimal schema for a system life cycle that can be tailored to the specific settings in each domain. Each setting can be thought of as a theater for the "hand-off" of *system artifacts* with people in different organizational settings. There can be many other settings involved as well (maintenance, tech support, customization, etc.) but the essence of the problem can be seen with the simple distinction between the *development settings* and *usage settings*.

Software systems embed implicit contextual knowledge. Because of the knowledge-intensive nature of the software "handoff" across settings, a great deal of contextual knowledge needed for organizations to run effectively gets "compiled" into software systems. Only a very small part of this knowledge is ever articulated as formal requirements. Inevitably, much of this knowledge becomes hidden or implicit to practitioners. The handoff of software systems across different organization settings can become a magnet for many process and communication breakdowns

---

[3.] Terminology note: Another familiar term for some readers might be "environment." However, in some interpretations this term means aspects of the setting *outside* the system boundary, while in other interpretations it means specifically the hardware/software environment. Neither of these interpretations would match the intent of "setting" here, which would encompass both kinds of environments as well as other aspects.
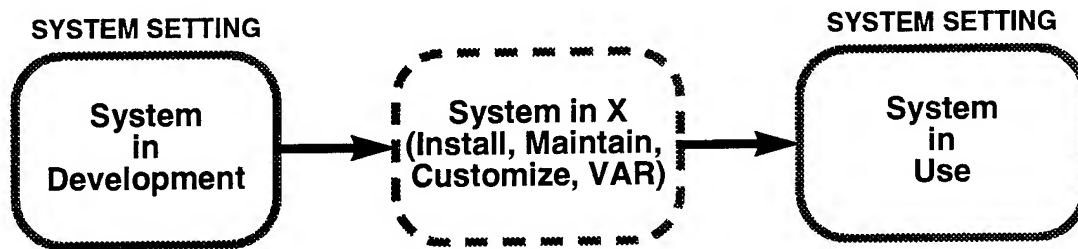
Exhibit 7. System Settings in a Minimal Domain-Specific System Life Cycle

because of this implicit contextual knowledge. Exhibit 2 shows some contextual layers surrounding domain functionality within a given setting.

When developers attempt to *reuse* software in new applications, these problems increase by an order of magnitude. When assumptions based on this contextual knowledge are built into systems and then transferred to new settings where the assumptions do not hold, many problems in predictability, quality and reliability result.

The Software "Value Chain"

Software settings are related in "enabling technology value chains". Development and maintenance of a software system is *not* a typical producer-consumer "food-chain" or "supply chain" as applied to the production and use of material goods. Software is a knowledge-intensive enterprise that serves as enabling technology in the customer's environment. The manufacturer of machine tools has a different relation to the factory in which the tools are used than the supplier of the raw materials which the factory processes into manufactured goods. In the same way, a developer of a software system interacts with the system users as a "tool developer" or "technology developer"
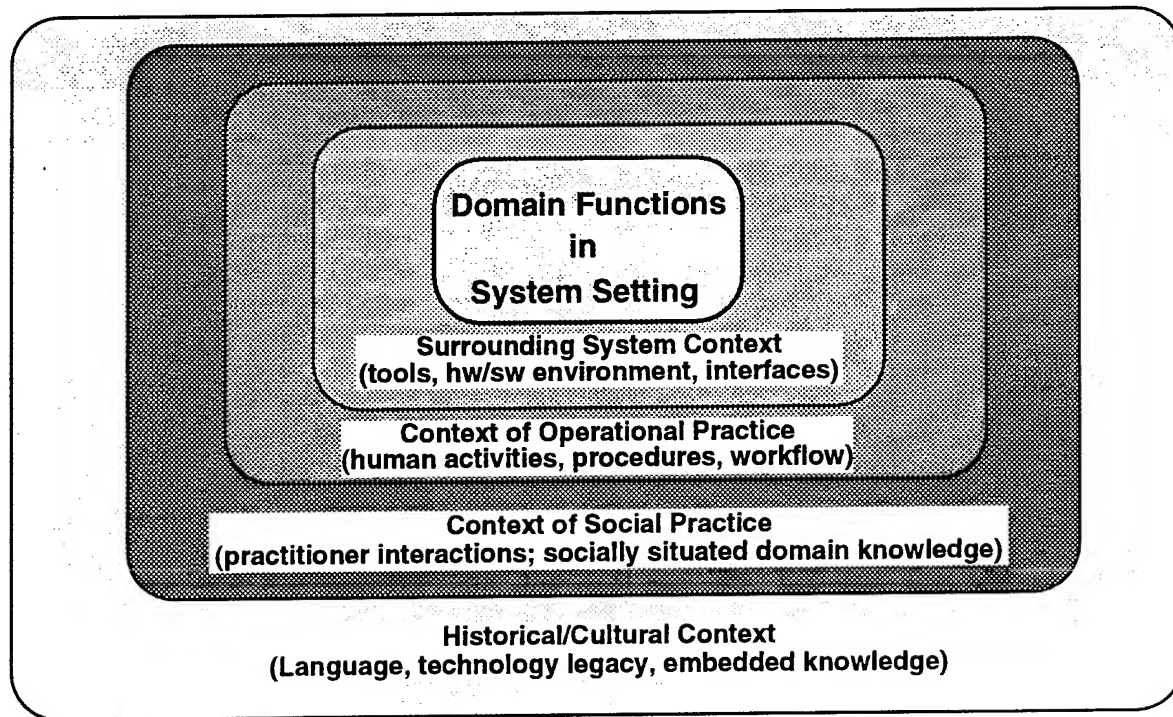
Exhibit 8. Contextual Layers Operative in Each Domain Setting

34

relationship. These enabling technology linkages can form more complex webs of relations, including chains and networks of technology providers and consumers.

The "customer" in the reuse context is generally not the same "customer" as in either the application or product context. Builders of reusable software are usually at a different level of the "software value chain" the software application or product developers, hence one step removed from those the host organization views as the "real customers." Because of this extended "value chain" the domain engineer has the option of designing an asset that is customized through design- or compile-time options exercised by the application engineer.

This creates problems in representing domain-specific software or systems knowledge. Representations used by system engineers almost always refer to the "operational environment," the world in which the implemented system will be executing and interacting with users. (We will refer to these informally as "system models.") Domain engineers may need to consider a variety of other settings, including the setting in which the application developers themselves are operating. In order to create conditions for an asset base within a given organizational context, it is often necessary to introduce a new layer in this "value network": domain engineers are providers to application engineers, who are providers to end-users.

Feature Binding Sites

One reason the notion of setting is so central to ODM has to do with the mutable nature of software. When we build functional or operational models of software systems, we are almost always implicitly placing ourselves in the context of the "system operation setting," the work environment where the software will execute on a computer and perform its real-world task, be that sensor control, embedded functions in hardware, or interacting with a human being performing some task. When we think of "system features" in the conventional sense, these usually map to some identifiable functions or capabilities of the application as it executes in this "end-user" or "operational" setting.

When we design software for reuse, however, we are often concerned with variations that are introduced in the development setting. In its simplest terms, this is a distinction between run-time and compile-time variables. A simple example would be the choice between tiled and overlapping window protocols in a window management system. Some systems hard-wire specific choices of tiled or overlapping windows. Others allow this to be specified in a configuration file read at start-up time. Still other systems might have a "preferences" switch that can be changed dynamically at run time by the user.

In comparing multiple systems, we sometimes find that a function supported at run-time by one system has been "hard-wired" in for another system. For domain modeling, we need terms and representations rich enough to capture these distinctions. Since domain models describe the commonality and variability across multiple systems, they must distinguish those combinations of features that must be *simultaneously satisfied* by a single system from combinations where each feature *individually* must be supported by some system derivable from the domain model. Essentially, the notion is that a choice between variant features can be "bound" or committed at various points within the software engineering life cycle.

In theory, one might believe that there is no reason to distinguish these different points because a system flexible enough to change behavior at run-time could simulate any variant that made its choice at compile time. It is tempting to assume that the latest (i.e., most "flexible") feature binding time is inherently the most "reusable", since in theory such a system can simulate the behavior of system variants with earlier binding times. But all engineers know this is an abstraction, because it ignores the many design and performance trade-offs that may be involved in selecting

35

the binding time for a given feature. For example, there may be systems where it is specifically *not* desirable for users to be able to switch from tiled to overlapping windows.

In ODM, the general term introduced is *feature binding site*. A system life cycle, as described above, consists of a network of system settings linked by the exchange of software artifacts. A typical characteristic of this exchange is that a software artifact created as a workproduct in the development setting is executed in the operational setting and effectively becomes process. A feature which shows up as a system capability in the operational setting has typically been "bound" by a series of design and implementation decisions in the development setting. Rather than introduce a fixed set of terms for "compile time," "run time," etc. we use the general term *feature binding site* to refer to any distinct point within a setting where system functionality or features can be actively determined, usually by some input from a person. ODM is not the only domain analysis method to recognize the importance of feature binding sites. However, it is the only method that has generalized the notion of domain settings so that these sites can be described precisely and comparatively.

Feature binding sites are points during a system life cycle at which feature variants may be selected. Depending on the setting and the nature of the feature being bound and the site where it is bound, the selection may be done programmatically, by software, or by human decision-making. Different systems in the same domain may make different choices with regard to where feature variants are bound. Some systems will have sites that do not occur elsewhere. Some will provide multiple options. A system usage or operations setting in particular will often contain multiple feature binding sites. For example, user interface-oriented software will typically bind features at a number of distinguish binding sites: system install time, session start-up time, tailorability through preferences menus, dynamic defaults, explicit commands, "next-operation-only" overrides, etc.

Conventional system modeling representations (even object-oriented representations) may have inadequate means for distinguishing a system version that binds a choice at implementation time versus at run time. A run-time operation in one system may appear as a design decision in another. At a minimum, representations of commonality and variability should have sufficient semantic expressiveness to distinguish these cases, and the domain modeling process must be able to utilize these representations effectively.

This also creates the possibility for creative exploration of *shifts* of features across binding sites, one of many powerful innovation techniques incorporated into the method. In the interpretive modeling process, modelers document connections between the same logical feature as it occurs at different feature binding sites. In a final, innovative transformation phase of domain modeling, feature variants can be shifted within and across contexts to different binding sites. For example, a simulation environment used by integration test engineers might be used by system operators to capture scripts and macros to build new reusable assets codifying their specific work practices. Opportunities for generative reuse are often revealed by such contextual shifts.

Innovation and the Archetypal Domain

One strength of ODM is that innovation literally falls out of the process. You will get innovative ideas for novel designs, *feature combinations*, or applications for domain functionality almost every time you discover a previously hidden contextual assumption in your exemplars. In effect, the *domain modeler* lives out in advance some of the contextual adaptation process that would usually be deferred until using a component in a new setting. These discoveries are fun and generate a lot of excitement and energy. They help to move the process along; in fact, some of the "methodicalness" is there to discipline or "damp down" innovative discoveries so that they are manageable and do not swamp the process as a whole.

While some innovative insights are a natural by-product of comparative modeling, others result from specific formal activities. Modeling shifts of features across binding sites is one example of a repertoire of model transformation techniques employed within ODM that can operate directly on formal models of domain semantics.

This purposeful exploration of domain innovations takes place at a critical juncture in the ODM process, precisely at the transition from the descriptive modeling phase to the prescriptive asset base engineering phase. The efficacy of the process depends on all the core elements previously described: explicit scoping and clear boundaries established for the domain; thorough descriptive modeling of the commonality and range of variability across a specified set of exemplars; consideration of the specific stakeholder interests and settings for the domain. Yet it also requires a momentary letting go of the strict descriptive tether, a free space between the empirically driven descriptive modeling phase and the pragmatically driven prescriptive asset base engineering phase.

Use of systematic discovery techniques on descriptive models tends to reveal incremental and fine-grained innovations, such as novel feature combinations, that are based on precedented features. In addition, the process of finding out why certain feature combinations were never provided often elicits subtle contextual information missed in earlier modeling. Innovation modeling thus serves as a model validation step, in addition to generating useful innovations that may (or may not) be later included in the prescriptive model.

The Reuse Paradox Revisited

The goal of these formal modeling techniques, still an area of active research, is more than just exploration of innovative possibilities for novel features and feature combinations, although this is a valuable side-benefit. Their essential role is to achieve *closure* of the domain feature space. Without this step, a domain model would reflect purely accidental descriptive boundaries of the representative set, or pragmatic choices of features oriented towards anticipated customers of the asset base. Such a model would not necessarily prove robust and evolvable over time. This is why this is an essential step in the ODM process, even for domains where feature innovation per se is not an important motivator for domain engineering. The goal is to discover the "archetypal" domain model approximated by the descriptive process.

To understand why this is an essential step in domain engineering, suppose that we completed the process with a scrupulously descriptive model, capturing the commonality and variability across a specific set of exemplar systems. There will inevitably be gaps in such a model, revealed by attempts to apply the model to new exemplars. As these gaps are discovered it would certainly be possible to treat each new application as a new exemplar and to re-play the process, updating the various models as appropriate. Each new application will inevitably approach domain functionality from a new viewpoint; if the only recourse is to iterate and enrich the exemplar set, in what way can we claim the domain model has predictive power?

In theory, one could have started from a single exemplar and expanded the model incrementally with each new application required. This suggests a successive refinement process where new exemplars are "thrown in the pot" until the resulting domain model stays unperturbed by new information. But this ignores the value added by parallel examination and analysis of multiple exemplars, a core aspect of the ODM method. On reflection, this can be seen to be the very crux of the matter in domain engineering.

> A good domain model can tell us not just what we *want*, but what we *need*, in terms of functionality within its scope.

We do not want to produce a domain model that is merely the "union" of a set of multiple applications in the domain. Such a model could be produced merely by "parallelizing" a conventional requirements analysis or systems analysis process; it would be a kind of "variability engineering" where no explicit domain concept would be necessary. If the model were merely the "grab-bag superset" of all conceivable feature variants, from which applications just pulled in the profile needed, the model would arguably offer little substantive domain knowledge to the developer.

The fallacy in this picture is the assumption that the domain model must exactly accommodate the stated requirements of a given application to be usefully predictive. This misses much of the point of the domain engineering endeavor as a whole; for the predictive value of a model lies precisely in its "informative mismatches." A domain model should partition domain functionality in ways that will generally tend to be optimal across a well-specified range of applications with particular characteristics. If a new application requests features outside the scope of the domain, the domain definition will indicate this. For features within the domain scope, we want the model to reveal features we might not have thought of, or warn us that certain combinations will prove to be more trouble than they're worth.

In the end, we arrive back at the search for the "right abstraction" that led us on the Reuse Grail Quest at the beginning of this section. While ODM may appear to fly in the face of this technical idealism, we actually believe the process has the best chance of systematically guiding us to the discoveries of these abstractions. The difference is that, rather than relying solely on innate talent and inspiration to guide us, we ground our conceptual understanding of the domain in selected empirical data, reflect on and challenge the contextual assumptions we bring to the process, and, ideally, emerge with domains that are durable, sharable and serendipitously useful in both expected and unexpected ways.

Part II: ODM Processes and Products

This part of the guidebook describes the core ODM process model and work products. It builds on and refers back to the core concepts introduced in Section 3. This part of the guidebook is structured as a reference to be used in working through the method in practice.

This introductory portion provides a brief overview of how the ODM process is organized, describes how Part II and its sections are organized to reflect the process structure, defines the conventions used to present information within the sections, and provides guidance in how to read and interpret the information presented.

Overview of ODM Process Model

In general, the ODM process model is described in accordance with the notations and conventions associated with the Unisys STARS Process Definition Process [Klin95a]. The primary notations used in this guidebook are process trees and IDEF$_0$ diagrams.

The process model description offers considerable guidance for performing the core ODM life cycle activities:

- selecting and defining a domain of focus;

- modeling the range of potential variability within the scope of the selected domain; and

- engineering an asset base that satisfies some subset of the domain variability, based on the needs of specific target customers.

ODM Process Tree

The process model is organized hierarchically. The full model hierarchy is shown as a process tree in Exhibit 9. Each node in this tree represents an ODM process. The nodes below a given process represent the subprocesses that are carried out in performing that process. The following terminology is used in referring to processes at each decomposition level within the process tree:

- *Life cycle*: The top level, or "root" node, of the process tree (i.e., *Domain Engineering*), representing the overall ODM domain engineering life cycle.

- *Phase*: One of the three major components of the ODM life cycle: *Plan Domain*, *Model Domain*, and *Engineer Asset Base*.

- *Sub-phase*: A set of tasks that collectively perform a coherent higher-level function within a phase; e.g., *Describe Domain* within the *Model Domain* phase.

- *Task*: A low-level set of activities where detailed ODM work is performed and workproducts are produced; e.g., *Model Concepts* within the *Describe Domain* sub-phase.

In general, the process tree provides a way of understanding and classifying activities at varying levels of detail. The tree can be viewed and used in several ways, each important:

- As a *primary* sequence: i.e., a depth-first "walk" of the tree represents the basic sequence of activities that tells the clearest story about the method as a whole process. However, one should not read too rigidly sequential a story from the process tree. Alternative sequences are
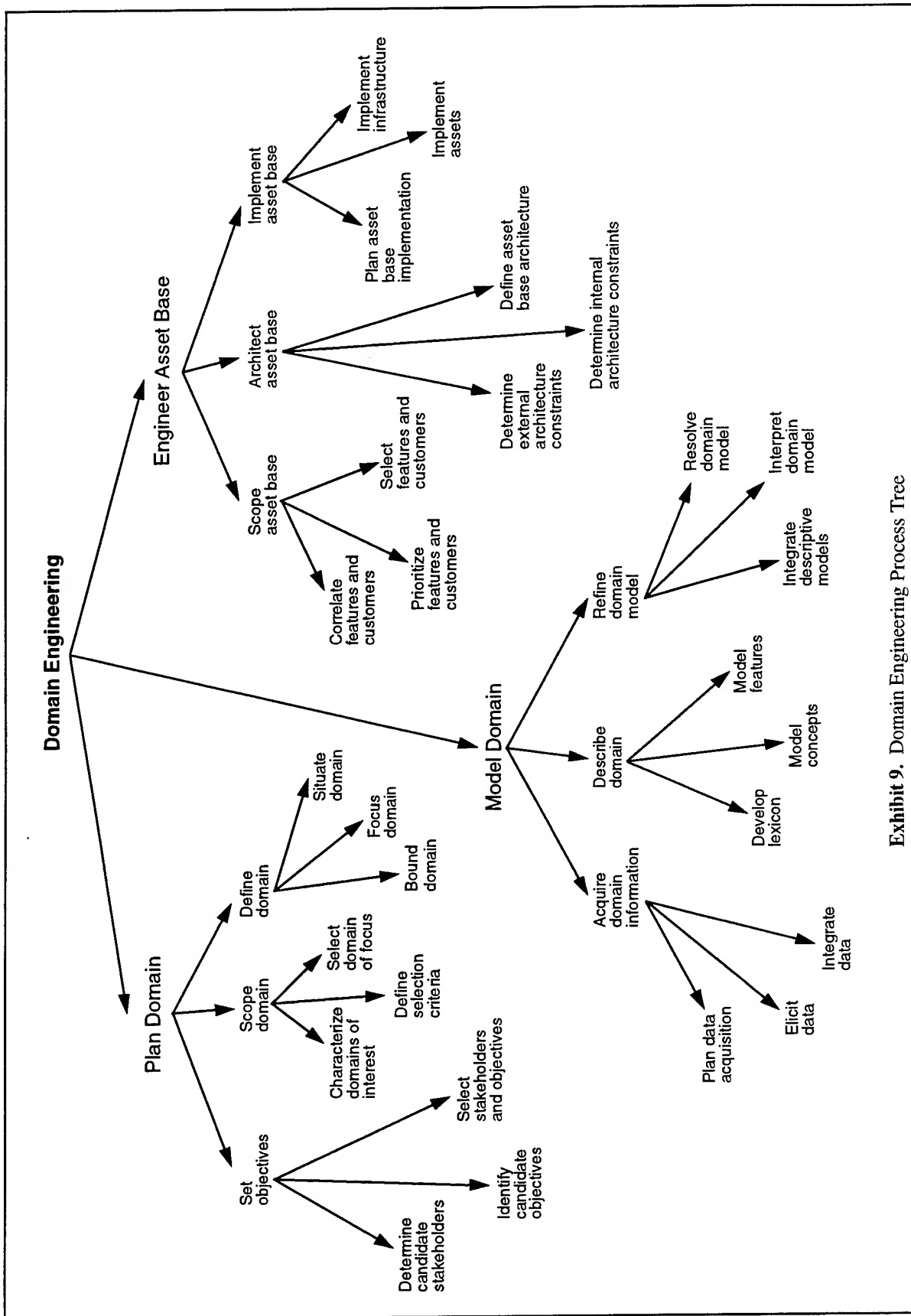
Exhibit 9. Domain Engineering Process Tree

possible, and iteration and parallelism are often valid strategies.

- As a taxonomy: i.e., nodes in the tree represent categories of processes and lower levels represent more specific processes in that category. This view helps underscore the fact that processes in ODM can be performed in a variety of specific sequences, and at different scales of activity. The same basic process structure might be performed in the course of a few meetings or several months' effort.

- As a dependency graph: i.e., the clustering of particular processes within the tree tells you something about the degree of inter-dependency between the processes.

Supporting Methods and Layers

Although ODM encompasses all of domain engineering, the core method offers prescriptive and detailed guidance only within a relatively narrow scope, focusing on activities that are unique to *domain* engineering and for which ODM offers a unique or distinctive approach. Other activities that are under the umbrella of the ODM life cycle because they are needed for domain engineering, but not unique to it, or for which there are a number of valid methodological options from which to choose, are relegated to the category of *supporting methods*.

The scope of the core ODM method is further constrained to primarily address activities that are *necessary* for each of the major life cycle activities above. A number of value-added capabilities could be incorporated into ODM to address particular needs, but some organizations that do not have those needs could find the extra capabilities burdensome. Thus, ODM relegates such capabilities into a set of process *layers* that can optionally be selected by organizations that require them. Exhibit 10 shows how ODM has been designed in terms of these layers and supporting methods.

Exhibit 10. ODM Layers and Supporting Methods

41

would make strategic sense. Its focus is on a hypothetical organization, Persona, Inc., and the domain engineering examples are cast in the context of this organization.

- Initiating a real domain engineering project in the Outliner domain in one of our own organizations. By association, this has lent a strong sense of realism to the example effort.

The scenario is presented in full detail in Section 5.0, "Plan Domain," and excerpts of the example materials are scattered throughout the guidebook, most prominently in the *Plan Domain* phase. In many cases, these examples take the form of sample workproducts, whereas in other cases, they are presented in text form in special "example" paragraphs. In addition to the excerpts included in this guidebook, we intend to produce a integrated package of expanded example material that will be available separately. Some or all of this material may become accessible via the World Wide Web.

> *Disclaimer.* Any resemblance, in the portions of this document indicated as example material, to existing organizations, individuals, products, or technologies is purely incidental and is not to be construed as making any statements of fact. Any analysis of real products performed under this effort was done purely for the purpose of this example and does not constitute a formal review. We make no claim or warranty about the accuracy of any of this data.

Other workproduct-related material in the guidebook includes:

- *Worksheet templates:* Templates are provided in Appendix C which suggest particular worksheets and associated formats to be used in constructing specific workproducts. Until there is a better level of automated support for these processes each project that wishes to use these templates will need to adapt and tailor them manually.

- *Starter lists:* These are suggested content, generally intended to spark modelers' own discussion of the content relevant for their project. For example, the guidebook offers some sample criteria to apply in selecting promising domains for software reuse. The lists are not exhaustive nor mandatory.

Part II Structure

Part II is organized hierarchically, reflecting the structure of the ODM process model. The ODM life cycle is described in detail in Section 4. The *Plan Domain, Model Domain,* and *Engineer Asset Base* phases are described in Sections 5, 6, and 7, respectively. The sub-phases and tasks within each of the phases are each described in individual subsections organized hierarchically within Sections 5, 6, and 7.

The phase, sub-phase, and task sections each include a process tree diagram for the current phase, with a shaded area showing which portion of the phase is described by the section. These diagrams not only show the scope of each section, but also serve as recurring roadmaps to help readers determine their "location" within the process model. For example, Exhibit 11 below shows the diagram from Section 5.1, which describes the *Set Objectives* sub-phase.

The process trees present a simplified view of the more detailed structure of the process model, which is represented in $IDEF_0$ diagrams. The life cycle, phase, and sub-phase sections each include an $IDEF_0$ diagram showing the information flows among the processes at the next lower level. For example, each sub-phase section includes an $IDEF_0$ diagram showing the information flows among the tasks in that sub-phase. Appendix A includes the entire ODM $IDEF_0$ process

43
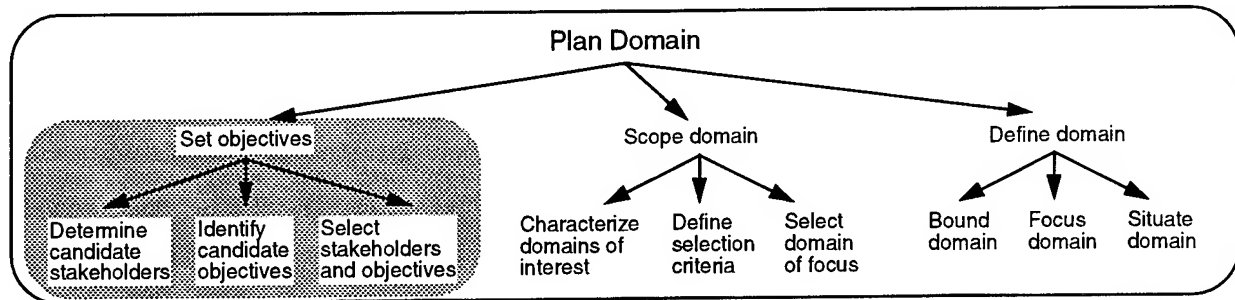
Exhibit 11. Example Process Tree Embedded within a Process Description Section

model. If you are unfamiliar with IDEF$_0$, please consult the appendix for a brief introduction to the IDEF$_0$ notation.

The general approach taken within the process description sections is to present information at the lowest section level at which it applies, while minimizing redundancy across sections. For example, low-level details about ODM activities and workproducts are presented in the task sections, whereas information about how the tasks within a given sub-phase are performed over time (e.g., sequencing considerations) is presented in the sub-phase section. The higher level sections also discuss the more general, conceptual, and strategic considerations involved in applying ODM.

Section Structure

The life cycle section, Section 4, provides an overview of the entire ODM process. It places domain engineering within the context of other broad activities, including general software and system development and overall reuse planning and adoption for the organization. It also introduces the three main phases and the major sequencing options and issues involving these phases.

The phase and sub-phase sections each include the following information:

- The introductory text provides context for the process described in terms of the overall life cycle and previous and subsequent processes. The overall objectives and benefits of the process are described, as well as (where appropriate) some of the key challenges in carrying out the process in the domain engineering context.

- **Approach**: The distinctive aspects of the ODM approach in meeting the objectives of the process and addressing the challenges described above. Key concepts relevant to the process are introduced here, as are examples to help clarify the concepts.

- **Results**: Describes the resulting workproducts and other outcomes of the process and the potential uses of these results.

- **Process**: An overview of what the process does and, to whatever extent is necessary, how it does it (emphasizing data flows and interactions among the lower level processes, rather than activities that occur within those processes, since they are described in detail in subsequent sections). The description may also introduce concepts needed to understand the lower level processes.

- **Guidelines**: Guidelines regarding performance of the process, focusing primarily on the sequence in which lower level processes can be performed. In general, ODM processes need not be performed in a particular sequence. Issues addressed here might include:

  — Different orders in which the processes may be carried out.

— Iteration: Lower level processes may be repeated in a "looping" cycle which has some criteria for termination.

— Consequences of skipping steps; e.g., starting the sequence in the middle, or terminating the sequence before the end.

— Parallelism: This can take various forms: e.g., multiple teams work different lower-level processes in parallel, with possible periodic hand-offs of information; or one group performs the processes in a highly interleaved way.

The Process and Sequencing discussions typically elaborate on the process tree and $IDEF_0$ diagrams included with the section or use them to illustrate points about the process.

The task sections (e.g., 5.1.1, *Determine Candidate Project Stakeholders*) constitute the bulk of the process model description. These sections describe the low-level ODM activities, workproducts, and information flows in detail and also offer tactical guidance for regulating and controlling the processes. The task sections are organized as into the following segments:

- The first paragraphs set the process context for the task (where have we come from, where are we going) and outline the task's key objectives and challenges, and the distinctive aspect of the task within the domain engineering life cycle.

- **Approach**: Describes the ODM approach to accomplishing the task objectives. Key concepts are introduced and illustrated where appropriate with examples related to the example domain.

- **Workproducts**: Key results of the task. The value of each workproduct is described within the context of the ongoing domain engineering life cycle.

The three segments above provide a good overview of the task for general comprehension. The remaining segments are targeted more directly to the practitioner performing the task:

- **When To Start**: A set of conditions that should be satisfied before the task can begin.

- **Inputs**: Information that the task accesses and manipulates in performing its function.

- **Controls**: Information that regulates or controls how the task is performed.

- **Activities**: Specific actions or steps to perform in carrying out the task and producing the workproducts. In general, the activities need not be performed strictly in the order they are listed, although this can vary significantly from task to task. An up-front introduction to the activities outlines whether they are best viewed as a sequence or as a "repertoire" of alternatives. This introductory text also calls out which supporting methods are referenced in the activity descriptions.

- **When to Stop**: A set of conditions for determining when the task is completed.

- **Guidelines**: Hints, suggestions, and criteria for performing the task effectively. Validation and verification criteria and techniques may also be provided here to determine the completeness, consistency, or quality of the workproducts. (These may also appear in the Activities sub-section when they are best described as distinct activities.)

The Inputs, Controls, and Workproducts directly reflect the inputs, controls, and outputs on the $IDEF_0$ diagrams.

The full set of Workproduct descriptions appearing in the task sections is collected for reference purposes in the ODM lexicon, Appendix B, in summary form. As noted above, templates for a selected set of ODM worksheets are included in Appendix C.

Examples are woven throughout the text, at whatever level they are deemed most useful. They are set off in distinct paragraphs with the key-word Example:

Presentation Conventions

A number of conventions were applied in writing the sections within this part to make the process model descriptions easier to read and understand. These conventions include:

- **Section subheadings**:

    All of the first-level section subheadings (Approach, etc.) are in bold on a line by themselves. For example:

    ### Approach

    These subheadings are also in a slightly larger font than regular text paragraphs.

- **Information under section subheadings**:

    — Approach, Results, and Process:

    The information under these subheadings is prose paragraphs, in whatever format is most appropriate for the material.

    — Workproducts:

    Each workproduct produced by the process is signified by a subheading on a line by itself in the following format:

    ■ DOMAIN MODEL

    Substructure within the workproducts is generally shown using bulleted and sub-bulleted items underneath the subheadings.

    — Inputs and Controls:

    These are presented as bulleted lists. Each item in the list includes the $IDEF_0$ data item name in an underlined run-in heading, followed by some explanation of how the item relates to the process. E.g.:

    - DOMAIN MODEL. This process uses the DOMAIN MODEL to . . .

    — Activities:

    Each activity associated with the process is signified by a subheading on a line by itself in the following format:

    ➤ Analyze Usability

    Under each of these subheadings, there may be subordinate subheadings representing subactivities. These subheadings look like:

Product Line Segmentation

— Guidelines, When to Start, When to Stop:

These are also presented as bulleted lists. Each item typically includes an underlined phrase (usually a run-in heading, but not always) that concisely summarizes the item, followed by explanatory text. E.g.:

- <u>Document model boundary issues/decisions</u>. Negotiating clear boundaries . . .

Under these subheadings, there may also be non-bulleted prose paragraphs providing sweeping or summary guidelines.

— Examples, Caveats, Notes, etc.:

These are special paragraphs, indented from the surrounding text, which provide example-related material, caveats, notes, or other special information. These usually begin with the word "Example," "Caveat," etc., in italics.

- **Typographical conventions:**

  — SMALL CAPS (WITH INITIAL LARGE CAPS): ODM workproducts (i.e., any workproduct produced directly by an ODM process) or suggested worksheets;

  — ALL SMALL CAPS: Other IDEF$_0$ data items (net inputs to the overall process);

  — *Italic With Initial Caps*: ODM process names (phase, sub-phase or task). Activities are generally referred to only within their own task description section, and with no special typographic conventions.

  — ***bold italic***: An instance of a key term in the ODM lexicon (usually this is reserved for the initial instance of the term within some cohesive portion of the guidebook, such as a section or group of sections). Even (or especially) when these are familiar terms to the reader (e.g., ***customer***) this convention is a signal that the term is being used specialized meaning within the ODM context.

  — *italic*: Emphasizes or highlights any term or phrase in running text.

Note that **bold regular** highlighting is used only in section headings, subheadings, or run-in headings within paragraphs (with the exception of this sentence).

Here is an example of the use of these conventions in context:

In the *Model Domain* phase of ODM modelers need to apply the skill of ***descriptive modeling*** as an attitude or mindset in producing the DOMAIN MODEL. Instead of deciding what *ought to be* in some particular system in the domain, they work with DOMAIN STAKEHOLDER KNOWLEDGE to model what *could be* in any system in the domain.

47

4.0 Domain Engineering Life Cycle

This section introduces the basic ODM process scenario and some of the key terminology used throughout the remaining sections in Part II. The basic sequence described in this document is a single-project scenario, producing a domain definition, domain model, and domain assets for a single domain of focus. Note that in some projects, it may not be necessary to complete this full scenario. Some possible ways of tailoring this basic sequence to meet specific project needs are discussed in Section 10.

The Domain Engineering Project Context

Domain engineering involves the creation of new technological and human systems for managing domains within some ORGANIZATION CONTEXT, potentially transforming both the organizations and technical systems within that context. The process tree depicted in Exhibit 9 (in the introduction to Part II above) shows the scope of a domain engineering project in terms of the processes involved. In the following descriptions, this overall project scope is called the *domain engineering life cycle*. This life cycle as presented here reflects the ODM approach to domain engineering, but is valid to some degree for domain engineering in general.

Domain engineering takes place in the broader context of a *reuse program*, which is described more fully in the STARS Conceptual Framework for Reuse Processes (CFRP) model [CFRP93a]. The CFRP outlines a taxonomy of reuse processes intended to span all reuse-specific aspects of reuse-based software development. The CFRP model (shown at a high level in Exhibit 12) is partitioned into two major submodels, known as *idioms*: Reuse Management (planning, enactment and learning) and Reuse Engineering (creating, managing and utilizing reusable software assets).
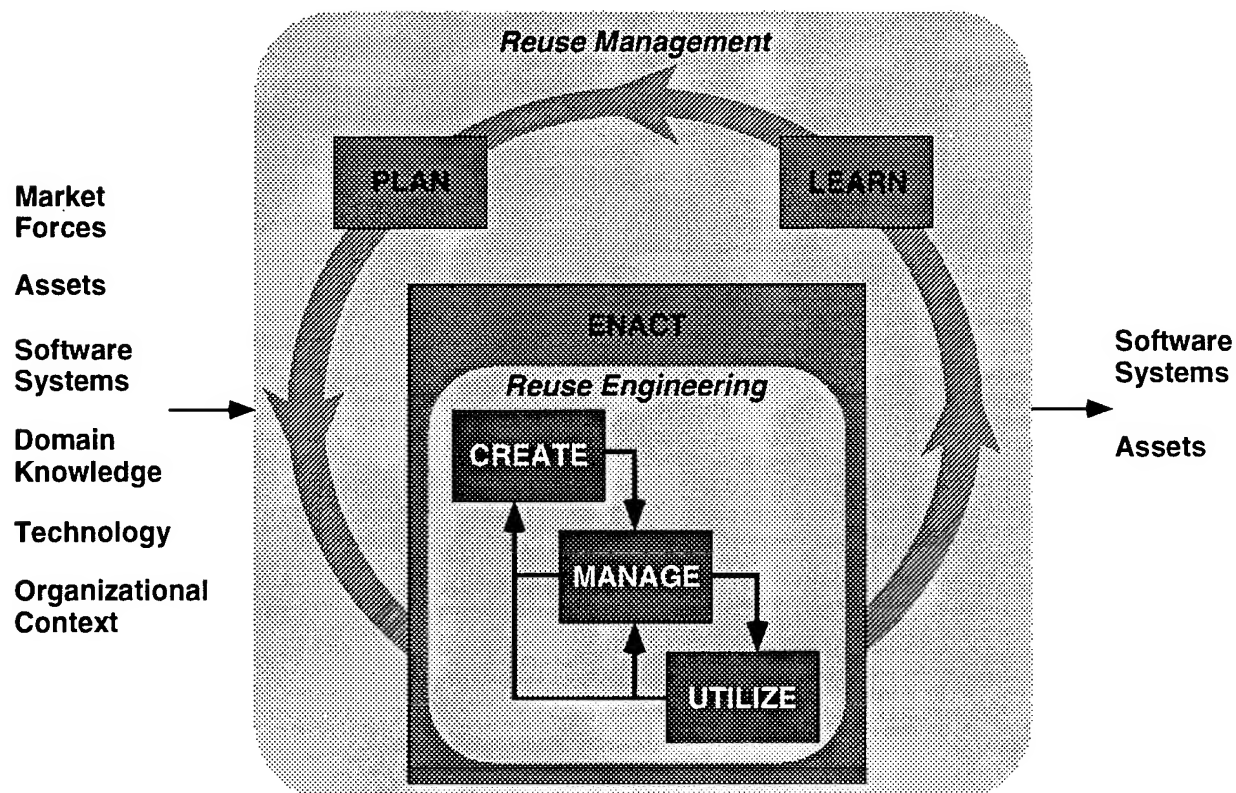
Exhibit 12. STARS Conceptual Framework for Reuse Processes (CFRP)

In CFRP terms, ODM focuses primarily on the Asset Creation (i.e., domain engineering) process family of the Reuse Engineering idiom. ODM's *Plan Domain* phase can be considered a specialization of the processes called out in the CFRP's Reuse Planning family of processes (with the exception of Infrastructure Planning and Project Planning, which are dealt with, respectively, in the Supporting Methods described in Section 8 and the Guidelines for Applying ODM in Section 10). CFRP Learning processes are encapsulated in a separable Learning layer of ODM, described in Section 9.

Domain engineering does not include the ongoing management of the domain model and asset base, nor utilization of the asset base by application development projects (i.e., the CFRP Asset Management and Asset Utilization processes, respectively). Relationships between the domain engineering project and other efforts, such as system reengineering projects or planned new products, must be carefully considered in planning and managing the overall reuse program. Discussing these relationships in detail is beyond the scope of this document.

Exhibit 13 shows the IDEF$_0$ context diagram for an ODM domain engineering project initiated within some ORGANIZATION CONTEXT. Depending on the nature of the *domain engineering initiative* and funding for the project, the project context can be:

- a single organization or a division of a larger organization, such as a corporate or university research and development facility, a product-line division, or a system development group,

- multiple organizations, as typified by standards organizations and consortia with common interests in particular functional areas,

- a marketplace of varied competitor, partner and customer organizations, or

Exhibit 13. Domain Engineering IDEF$_0$ Context Diagram

50
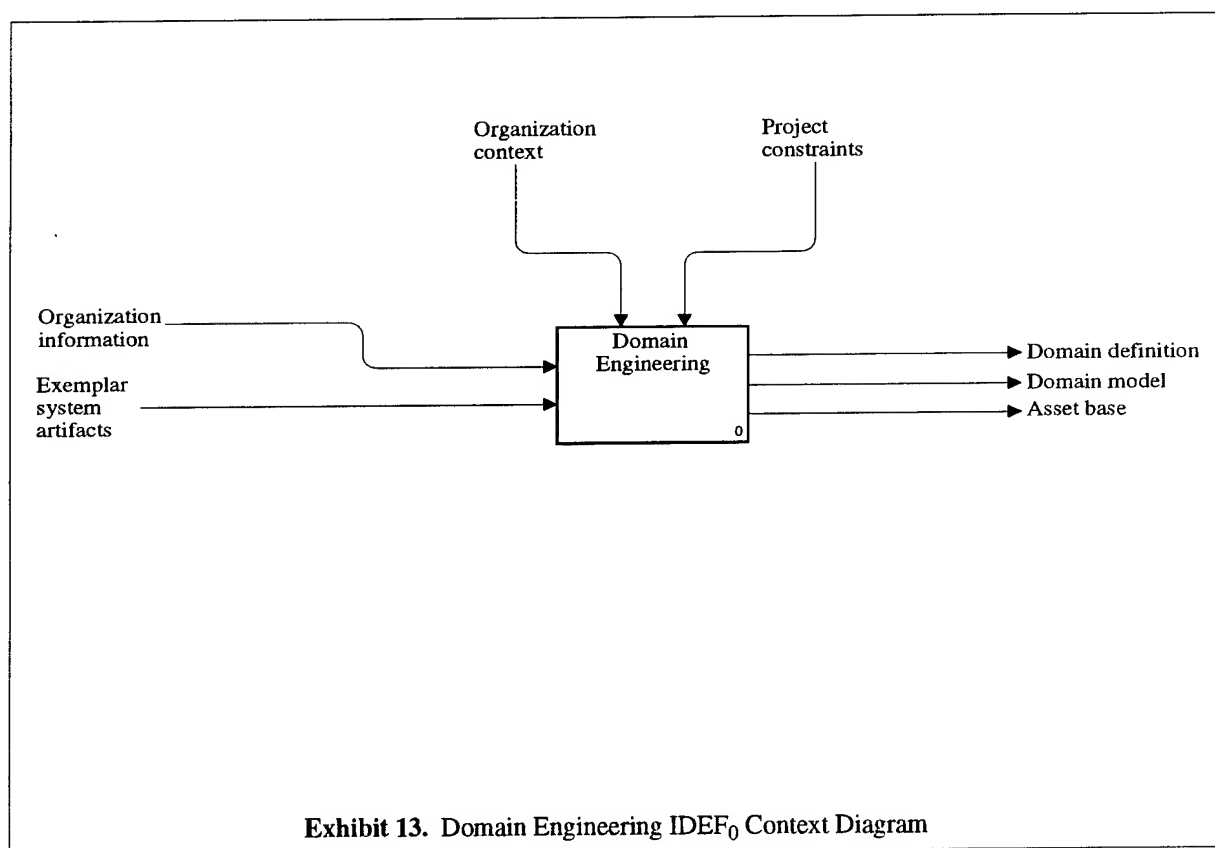
- a technical community.

These stakeholder interests and relations form the *stakeholder context* for the domain engineering project. Descriptions may informally refer to this context as "the organization" for convenience.

The two primary inputs to *Domain Engineering* are ORGANIZATION INFORMATION and EXEMPLAR SYSTEM ARTIFACTS. ORGANIZATION INFORMATION includes:

- Human domain expertise,

- Textbooks, and

- Industry sources of information.

EXEMPLAR SYSTEM ARTIFACTS include the various artifacts pertaining to exemplar systems available for study. Following are a few examples of typical artifacts:

- Source and/or object code,

- Requirements, architecture, and design specifications,

- Test plans, test cases, and results,

- System development process models,

- Budgets and schedules, and

- Maintenance histories.

The project produces three key results: a DOMAIN DEFINITION, a DOMAIN MODEL, and an ASSET BASE. An *asset* may be a software component, a generative tool, a template for a design document, or any workproduct of the software life cycle specifically engineered for reuse within a domain-specific scope of applicability. The ASSET BASE is the full set of ASSETS for a domain, together with the ASSET BASE ARCHITECTURE that integrates the assets and ASSET BASE INFRA-STRUCTURE required for the domain.

Phases of Domain Engineering

Domain engineering consists of three main *phases*, as shown in Exhibit 14: *Plan Domain, Model Domain,* and *Engineer Asset Base*. These phases are called simply (Domain) Planning, (Domain) Modeling, and (Asset Base) Engineering in the following paragraphs for convenience.

The primary purpose of the *Plan Domain* phase is to set objectives and select and scope a domain for the project in alignment with overall organizational needs. Explicitly modeling the various stakeholders for the project in the DOMAIN STAKEHOLDER MODEL is a key task to ensure this alignment. PROJECT OBJECTIVES and a DOMAIN DEFINITION are the other key outputs of this phase. The Planning phase includes considerable definition work beyond simple selection of the domain, to ensure that the domain is appropriate for detailed modeling. Obtaining commitment of the various stakeholders involved is also a key task of this phase.

The primary purpose of the *Model Domain* phase is to produce a DOMAIN MODEL for the selected domain, based on the DOMAIN DEFINITION produced in the Planning phase. The key role of the DOMAIN MODEL is to describe common and variant features of systems within the domain, with rationale for the variations. The DOMAIN MODEL is subsequently used in the Engineering phase as

Exhibit 14. Domain Engineering IDEF$_0$ Diagram
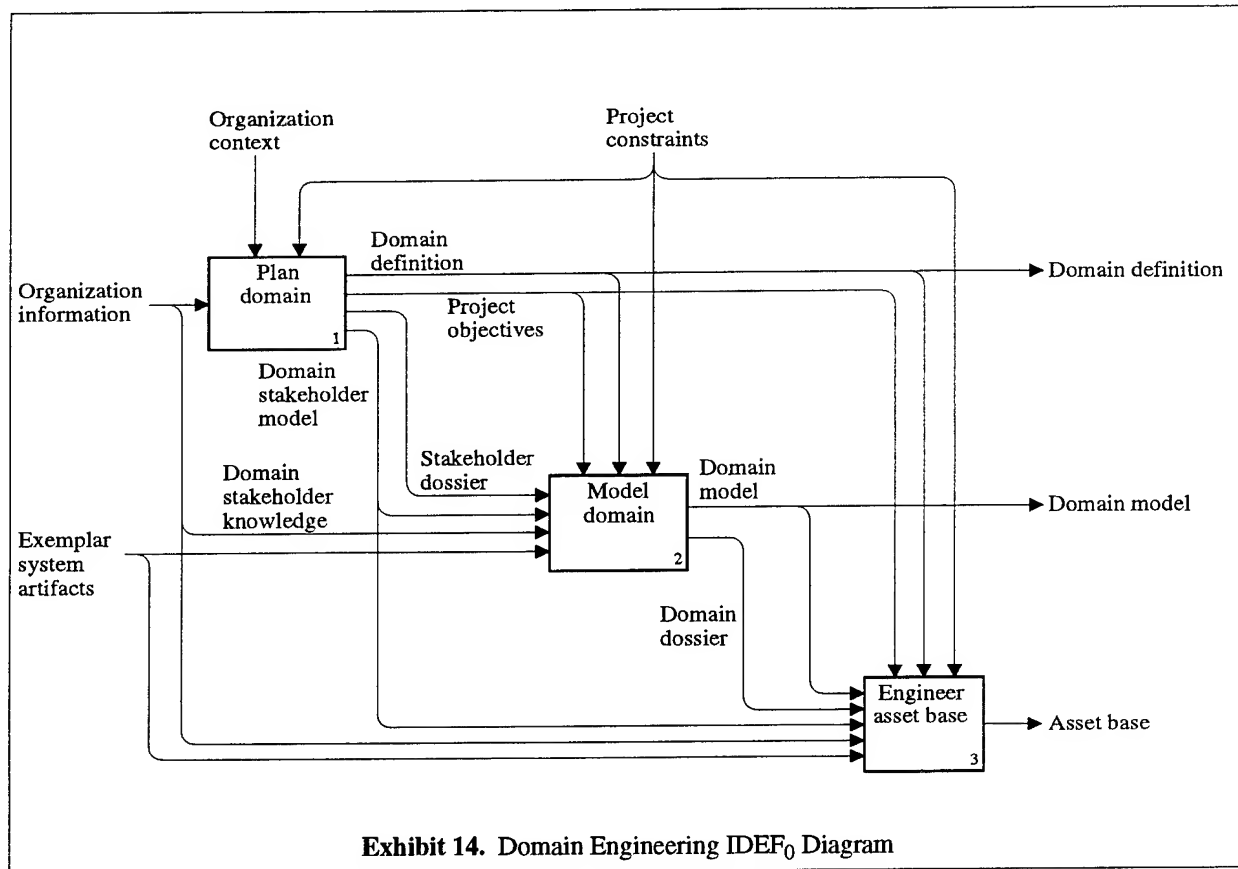
a basis for selecting the range of variability to be supported by ASSETS in the ASSET BASE. A secondary product of this phase is the DOMAIN DOSSIER which documents the specific *information sources* used as a basis for modeling. The DOMAIN DOSSIER may also be used during the Engineering phase to trace legacy artifacts that are candidates for reengineering into reusable assets; or to identify constraints in systems into which assets may be migrated.

The primary purpose of the *Engineer Asset Base* phase is to scope, architect, and implement an ASSET BASE that supports a subset of the total range of variability encompassed by the DOMAIN MODEL, a subset that addresses the domain-specific requirements of a specific set of customers. The key benefit of the ASSET BASE produced in this phase is that it achieves an overall economy in the cost of developing systems in the domain, when viewed from the perspective of the asset base customers as a whole. The key challenge of this phase is identifying an appropriate market of customers and the features required to support this market, to make the ASSET BASE a viable and ongoing structure for achieving increasing levels of reuse. An additional challenge in the Engineering phase is to manage the potentially explosive variability in the domain to eventually produce tractable specifications for implementors to create an ASSET BASE INFRASTRUCTURE and individual ASSETS.

Readers may be surprised that the first phase is "merely" about planning activities, yet it is elaborated at the same level of detail in the tree as the later phases. Should planning the domain engineering project take fully one third of the overall time and effort? No; the first phase should take the least time, as one would expect. This phase has been placed under a "process magnifying glass" for these reasons:

- Analogous to "upstream" activities in software development, potential mistakes made in the early part of a domain engineering project can be the most costly (and the most unnecessary).

This is particularly true in domain engineering, where planning activities can become the basis for multiple modeling phases for different domains. Extra attention and care taken at the front end of the process will therefore have a relatively high impact on the overall success of the project. The process model is designed to facilitate this risk reduction strategy.

- People learning about domain engineering as they go along will be least familiar with key concepts and skills at the outset of the process, where, as stated above, most care is needed. Outside facilitation or consulting is a good risk reduction strategy to compensate for this. However, the process model is designed to be as robust as possible when used by a project team that is "learning by doing." This requires closer attention at the front end of the life cycle.

Note that, in general, the potential variability of the life cycle from one organization to the next is greatest at the front and back ends. At the front, where core processes interact with many organization-specific and strategic elements; at the back, because increasing technology decisions and commitments lead to activities beyond the scope of the core process.

Incremental Results of Domain Engineering

The description of ODM in this Guidebook is designed to suggest a certain "primary sequence" of activities reflected in the top-level *Plan Domain*, *Model Domain*, and *Engineer Asset Base* phases. Alternative sequences are possible, though not addressed directly by the process model described herein. One way of viewing the motivation for alternative sequences is in terms of potential alternative uses of workproducts. Such alternative uses, and their implications on the life cycle, are listed below.

- The PROJECT STAKEHOLDER MODEL represents, in effect, a strategic analysis for the stakeholder context of the project. This workproduct could be of immediate use in broader reuse adoption efforts, strategic planning and alliance-building, and, of course, for follow-on domain engineering efforts for new domains.

- The DOMAINS OF INTEREST serves in the core process as a precursor to domain selection. However, it also represents a "domain-oriented scan," as it were, for the stakeholder context. This kind of analysis could be of great value in defining and re-envisioning the core competencies of an organization to support organization redesign efforts like business process engineering, decisions about acquisitions, out-sourcing and competitive advantage, or prioritizing efforts to retain or acquire critical sources of expertise for the firm.

- The DOMAIN DEFINITION can be used by:

  — domain planners who wish to understand the potential overlap between distinct domains;

  — domain modelers who want to do descriptive modeling of representative systems in the domain.

  Updated to represent the scope of the final ASSET BASE, the DOMAIN DEFINITION is also useful to:

  — component developers and asset managers;

  — potential utilizers, who wish to determine whether or not a component they seek belongs within the scope of the ASSET BASE; or

  — Submitters of draft components to the ASSET BASE, who wish to distinguish assets absent due to deliberate exclusion from those absent due to gaps in coverage that could

properly be addressed with a new asset.

- The DOMAIN DOSSIER can serve as a valuable consolidated source of systems expertise for the targeted domain. For those exemplar systems studied in detail, it can serve as a source of reverse engineering knowledge and documented rationale for key design decisions. It can be used directly in situations where opportunistic reuse, best practices dissemination, or adaptation from examples are reasonable approaches to reuse. This may be appropriate not only in certain organizational cultures but also depending on the degree of maturity of knowledge in the domain. For a company where systematic reuse has become ingrained in overall engineering practice, maintaining a project dossier would be a routine way of maximizing the learning from innovative projects in new technical areas.

- You don't need to build an ASSET BASE to obtain value from the DOMAIN MODEL. The DOMAIN MODEL can be used in a number of engineering activities, including, but not limited to the *Engineer Asset Base* phase described in this guidebook. These uses include:

  — guide application developers directly, without the presence of an ASSET BASE (using a *model-based development* approach to improve the software development process);

  — train new personnel in codified knowledge for the domain;

  — provide domain practitioners a shared lexicon that reconciles variant terminology and synthesizes multiple information sources;

  — support creation of individual reusable assets.

  — hand off an "unpopulated" domain model directly to an asset management role; the asset manager would broker asset creation and utilization incrementally from the potential "market" defined by the web of stakeholders that have co-created the model.

- The *innovative features* that extend the purely DESCRIPTIVE MODELS into the final DOMAIN MODEL can also directly add value by revealing new kinds of capabilities and new opportunities for their application. In addition, because these features are introduced via systematic transformations from descriptive features, they are generally more tractable to implement than entirely new functionality would be.

This section has outlined a number of incremental results of the domain engineering process that should be kept in mind by project planners in order to maximize spin-off return on investment for the organization. Domain engineers may also want to keep these value-added possibilities clearly in mind at the conclusion of each significant stage in the process. Decisions to pursue such alternative approaches constitute tailoring of the ODM method. See Section 10 for further guidance on ODM tailoring.

The Domain Life Cycle

The domain engineering project life cycle described above extends the traditional notion of the life cycle for a single software development project, to encompass the notion of evolution of domain knowledge across a product line or family of systems, or across separate application efforts within a domain.

The domain engineering project life cycle, in turn, can be seen as one "moment" or episode within an even longer view: that of the life cycle of the domain itself. One view of this broader cycle has been described as the "domain life cycle" in an early paper [Simo91a].

Viewed as organized and shared bodies of knowledge, each domain goes through a typical evolution. A first phase is basically chaotic and disordered; a middle phase where some dominant paradigm emerges for coordinating concepts in the domain.

Some software controls or simulates systems that basically imitate or respond to natural scientific laws. But by far most systems reflect domains that are much more "cultural" and ephemeral in nature. For example, the domain of business accounting practices, while relatively stable and slow-moving from a technology standpoint, is hardly a domain of immutable and unchanging laws comparable to the domain of flight dynamics. And even in "scientific" domains, the progress of our knowledge means that new algorithms and techniques are constantly being discovered and refined.

Knowledge evolves in all domains, and domain engineering methods must provide support for this evolution over time. Nevertheless, there is a distinction between incremental learning and evolution and that suggested by domain engineering. We view domain engineering as a window of parallelism within a larger evolutionary pattern that might begin with a slower building up of knowledge through cases and examples. A domain model represents a "snapshot" derived by parallel analysis of multiple instances. Once a domain model has been established the incremental process begins again, enriching or possibly "decaying" the domain model structure. Eventually, the model may need to be substantively revised; or the domain may be absorbed into boundaries of neighboring domains or due to the obsolescence of the defining technology for the domain.
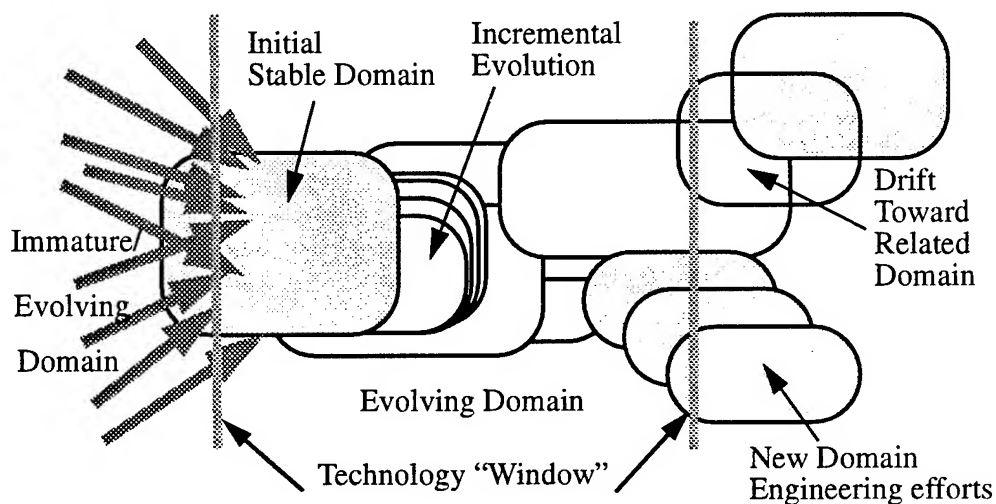
Exhibit 15. Domain Evolution

This means it is important to pick reasonable domains for application of domain engineering as an "accelerating function," and critical to establish realistic expectations for the results of the domain engineering effort. Domain engineering could not have stabilized window management systems much before the basic metaphors settled into common practice, nor could domain engineering help to salvage, say, a model of punch-card printing routines in the modern era. Biggerstaff's Rule of 3 says that a good domain for reuse should have at least three legacy systems (sufficient maturity) and at least three new systems should be anticipated (sufficient economic justification). Biggerstaff also introduced the notion of the "technology window" within which reuse and (more specifically for our purposes) domain engineering efforts would be cost-effective.

There are complementary approaches that address the "initial" stages of domain evolution where a focused domain engineering effort may be premature. For example, Scott Henninger has documented work with Union Pacific Railroad in evolving an "organizational memory" viewed specifically as a step preceding formal domain analysis [Henn96].

Punctuated Evolution. The ODM life cycle provides a common process structure for the domain engineering "moments" in the evolution of domains. In these moments, reflective study of multiple exemplars in parallel helps to create a cognitive leap forward to a qualitative reorganization of knowledge in the domain. Results of domain engineering will not remain static; domain definition, domain model and asset base will continue to evolve. As new assets are developed and placed into the asset base the "predictive power" of the domain model will be tested over time. After a period of evolutionary change, it may become necessary to initiate major reorganization or even follow-on domain engineering efforts in the same domain to adapt to changes in technology and practice. Some of these changes will themselves have been spurred by the original domain engineering results, collective reflection on the part of the domain's community of practitioners.

Guidelines

The ODM process structure allows for iteration and re-entry to accommodate both unforeseen risks and follow-on opportunities. This will most often occur as later extensions to the domain engineering project or subsequent projects within the same organization. Subsequent projects initiated in the same organizational context can select new domains of focus, reusing some workproducts from previous projects. By choosing different data as a basis for modeling, different domain models can be produced starting from a common domain definition. Similarly, a single domain model can provide the starting point for multiple asset base engineering efforts. Within each phase various workproducts and process checkpoints allow for efficient iteration if required.

It is also possible to manage projects with varying degrees of parallelism involved. This may allow for better use of resources, but will significantly increase both the technical and management complexity of the projects. Alternative strategies for project management of the overall domain engineering life cycle can be considered, including parallel, pipelined, or iterative strategies. In any sequence, however, a core principle of ODM is to maintain clarity about whether descriptive or prescriptive modeling is being performed, and to keep the resulting models separate.

5.0 Plan Domain

The *Plan Domain* phase of the ODM life cycle begins with initiation of a domain engineering project in some ORGANIZATION CONTEXT. The organization may have a mature reuse-based practice in place, or this project may be an initial step towards more systematic reuse. There may be a well-defined PROJECT CHARTER or only a vaguely understood mandate for the project.

The ODM process directs much of the early part of the *Plan Domain* phase to discovery and validation of an optimum set of objectives and a selected domain of focus for the project.
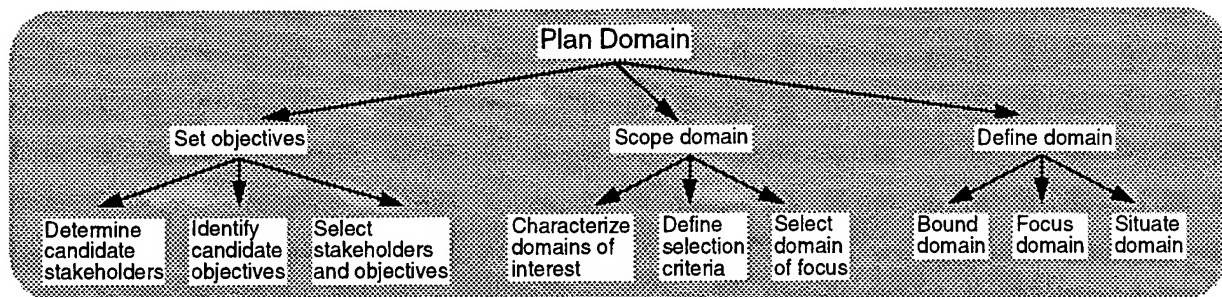
Exhibit 16. Plan Domain Process Tree

The primary purpose of the *Plan Domain* phase of ODM is to choose PROJECT OBJECTIVES and a domain of focus that are appropriately scoped and aligned with the broader strategic needs of the stakeholder organizations. This depends on establishing an explicit model of the ORGANIZATION CONTEXT for the project and the domain. Another overall goal of the *Plan Domain* phase is to assist stakeholders in applying a "domain-oriented" view of the business systems and relationships within the ORGANIZATION CONTEXT.

One key challenge in the *Plan Domain* phase is that stakeholders will often include multiple organizations or organization divisions, often with conflicting as well as coinciding interests. A domain engineering project involves negotiating a complex set of relations with customer application engineering groups, and other people who have a stake in the project.

In addition, domain engineering is still an immature discipline. The organization may have few standard ideas about reuse and domain engineering. There are likely to be significant misunderstandings (or, more properly, divergent understandings) of what constitutes a domain, what makes a domain a good choice for reuse, etc. Assessment and technology transfer needs to be an inherent element of the process.

Approach

Domain engineering can be initiated in a wide variety of organizational settings and can be in alignment with a variety of overall strategic objectives. An important part of the ODM approach to domain planning is to obtain agreement on an explicit set of PROJECT OBJECTIVES. To ground this discussion, we have introduced the example scenario above.

The *Plan Domain* phase includes a number of "upstream" tasks in domain engineering, many of which are not formally supported in other methods, including:

— determining PROJECT OBJECTIVES with reference to strategic interests documented in a PROJECT STAKEHOLDERS MODEL;

— making a strategic DOMAIN SELECTION from a candidate set of DOMAINS OF INTEREST;

— iterating between definition via exemplars and via defining rules, in order to uncover implicit contextual assumptions in the domain scope; and

— identifying neighboring domains via a rich set of relation types, including generalization, specialization and "analogy" domains.

To introduce some of the key elements of the Plan Domain phase, we will introduce the example scenario which will be carried through the remainder of the process model descriptions.

The Example Scenario

Example. Persona Technologies, Inc. (PTI) is a medium-sized (@75 employees) software firm specializing in user interface software technology for a variety of platforms, including Unix workstations, PCs (NT and Windows95), and Macintosh. PTI grew out of a service-based engineering consultancy, building customized interfaces for a variety of applications like medical accounting systems, law practice, and educational administration.

Several of PTI's large client bases were created by their working in partnership with other software firms who were working on database or transaction-intensive applications. While these jobs helped establish their revenue base and created several long-term relationships with clients, they have also often been frustrating because of the dynamic of "two customers": the end customer and the technology partner both make many requests for seemingly minor but actually quite labor-intensive fixes to interface code. More recently, partly in reaction to these entangled contracts, PTI has taken on some client applications entirely on their own. Here, though, lack of targeted vertical niche expertise within PTI has meant a large client-specific learning curve for each job, reducing their profitability.

PTI's general corporate challenge, at this point, is to either find a way to capitalize more intensively on their UI expertise, or to make a commitment to a complementary vertical application niche (e.g., medical information systems, legal services) and develop a compelling product in this area where their UI expertise can provide a compelling differentiator. The general mood within the firm is to stay with the UI focus.

Persona's head of marketing, Jack Perricone, has spent a good bit of time with end customers, both high-level managers in position to sign off on deals and typical end-users of their systems. As more and more systems migrate onto PC platforms, their products are compared more often to commercial PC software. Customers and users want familiar "look and feel" behavior from the interfaces built by Persona, not one-of-a-kind systems (no matter how elegant) that they need to learn from scratch. Thus, PTI can't afford to compete by building UI components that duplicate functionality available via COTS products like spreadsheets. Yet they do not want to slip into the role of integrators of standard packages. They prize their innovative prowess as developers of leading-edge UI technology and want to develop capabilities that will make their custom systems attractive. At the same time, PTI needs to be able to work comfortably across platforms, so they don't want to invest heavily in platform-specific technology.

There are several competing "technology visions" within the company, each of which offers a different kind of response to this overall business environment. The longest-established group in the company (in business for about ten years) began building interfaces on workstations and has a good deal of experience with X-Windows and MOTIF. This group has developed a MOTIF-based GUI-builder, called Maskara, but they have not really taken the step of productizing it; it is mostly used as an aid to service and contract-based work.

Some of the Maskara group's initial work was with government contractor organizations, and so they have developed some Ada expertise in-house. By now their work is migrating more into private-sector business, but there remains some interest in Ada95 in this group, an interest whetted by the Gnat compiler, the Ada-Motif bindings, and recent developments in Ada-Java integration. Other trends this group is tracking include emerging cross-platform interfaces such as OpenDoc, CORBA and FRESCO. They are tracking distributed object technologies such as CORBA and its potential impact on user interface system aspects. They are generally interested in engineering components and tools that can continue to work with their MOTIF-based technology but will be easy to retarget as these newer technologies mature.

A second group within the company predominantly consists of more recently hired employees (past three to four years). This group is focused on the opportunities provided by the World Wide Web (WWW) infrastructure. This group is led by an aggressive marketing team that is concerned with establishing a market presence quickly in the highly competitive and volatile Internet-based technology world. They believe they can leverage Persona's deeper experience with robust, large-scale, distributed UI applications into high-quality Web-based applications for their customers. They anticipate increasing pressure on interface-intensive systems to be "Web-savvy."

While the Web market has been led by a large population of enthusiasts new to computing, Persona's customers are by and large drawn from companies with mature, high-functionality systems in place. They want the quality of service they are accustomed to from other systems, but cast into Web-accessible intra and internet interfaces. Persona is therefore launching a new service division called Webonautics focused on the business of helping clients manage their information sources and present it over the web. They have discovered that much of this information is highly structured data that does not fit well with today's generation of Web browsers. Information exists in databases, knowledge bases, object bases and in highly complex structured documents. The information needs to be massaged and manipulated as it goes and comes over the Web to organization members.

Webonautics will offer combined consulting, analysis and engineering services for corporate clients. They will go internally to go into a client's site -- analyze the organization's morass of corporate data -- and quickly come up with a set of routines to filter and publish the data in effective, Web-compliant formats. Webonautics wants to be able to offer a number of "look and feels" (text-like, tree-like, "graphical", tables, spreadsheets, outlines) as well as a number of different architectural options for distribution. Depending on intra-organization connectivity constraints, the provision of various features can be client side (e.g., via Java applets or Javascript) or server side (e.g., with Perl/CGI). To do its job efficiently and cost competitively, Webonautics can't afford much custom code production whether it be Perl or Java.

In addition to the technology and tenure differences, there are different "cultures" within these two groups within Persona. The "Webonauts" (as they call themselves) have a hint of missionary zeal. They are convinced that the Web is the wave of the future and that it is essential for the company's survival that they be significant players in this market. The longer-established group is slightly disdainful of Web-based technology because of its functional limitations and "flatness" and believe that the Web "fashion craze" has been a bit over-hyped. Nevertheless they recognize the business opportunities.

The president of the company, Chris Connell, has pushed for the Webonautics direction, but now sees a potential rift within what is still a fairly small organization, and views this as a risk that should be addressed. Furthermore, the separation between the two groups means that efforts to create and capitalize on shared resources must be duplicated or have less impact on the company's bottom line.

Both groups are facing a combination of technical and business problems, including:

59

— an environment with vague, emergent, and rapidly evolving systems requirements that lead to numerous requests for fixes from customers;

— a growing customer base with long-term support contracts that are creating numerous configuration management hassles;

— a need to differentiate product and custom application offerings in terms of more than quality of service (most noticeable when absent) or efficiency in development (many of their jobs are fixed price; from the customers' perspective this differentiator translates purely to cost, and, again, is not very visible). They are looking for innovative approaches to user interface design that are particularly applicable to niche applications.

This is a general picture of the ORGANIZATION CONTEXT for the example scenario. The initiating event for the domain engineering project is the following:

The director of technology, Francesca Morrow, has attended some workshops on reuse. She begins to believe that there might be a way of addressing some of the company's issues using some techniques of systematic reuse. In a quick meeting with Connell, Morrow suggests that they fund a small-scale domain engineering pilot project to explore the potential for creating an internal library of reusable UI assets that could, at a minimum, lower the cost of developing turnkey applications for clients.

Connell's main interest is his concern over the growing separation of the two internal groups. He likes the idea of a project that might help re-establish a common direction, without appearing to impose a mandate to standardize from on high (a strategy that would not fly in his organization). On the other hand, he is concerned about a project that turns the groups too "inward," or has too much of an R&D feel. He tells Morrow to work with Jack Perricone to find an application that has some customer focus. With this constraint, Morrow gets approval for a small amount of funding to develop a more detailed proposal for the pilot. The informal verbal understanding reached at the conclusion of this brief meeting is the *sole* PROJECT CHARTER available at the start of the process described in *Plan Domain.*

Discussion

Given the Persona example scenario introduced above, let us consider the activities that initiative the *Plan Domain* phase. At this point in the story, there is no accepted consensus within the organization about reuse (systematic or otherwise). The project "opportunity" (for that is all it is at this point) is not couched in terms of a mandate to "perform domain engineering" but rather is motivated in terms of key strategic objectives for the company. Francesca Morrow is the "reuse advocate" at this point who is able to discern a pattern in the current environment that suggests the potential applicability of systematic reuse techniques. There is by no means firm commitment to the project within the organization. The PROJECT CHARTER does exist in that there is a basis for moving forward with some initial meetings. But there is no document, no clear statement of objectives, and certainly no sense yet of who the domain engineering team would be.

Why does the ODM process model begin at this radically "upstream" point in the process? There is a saying about engineering projects in general that "70% of the mistakes are made on the first day." While perhaps on the pessimistic side, the point applies here that in this early, informal part of the process, decisive relationships among stakeholders are established that can have a disproportionate impact on the eventual success of the initiative.
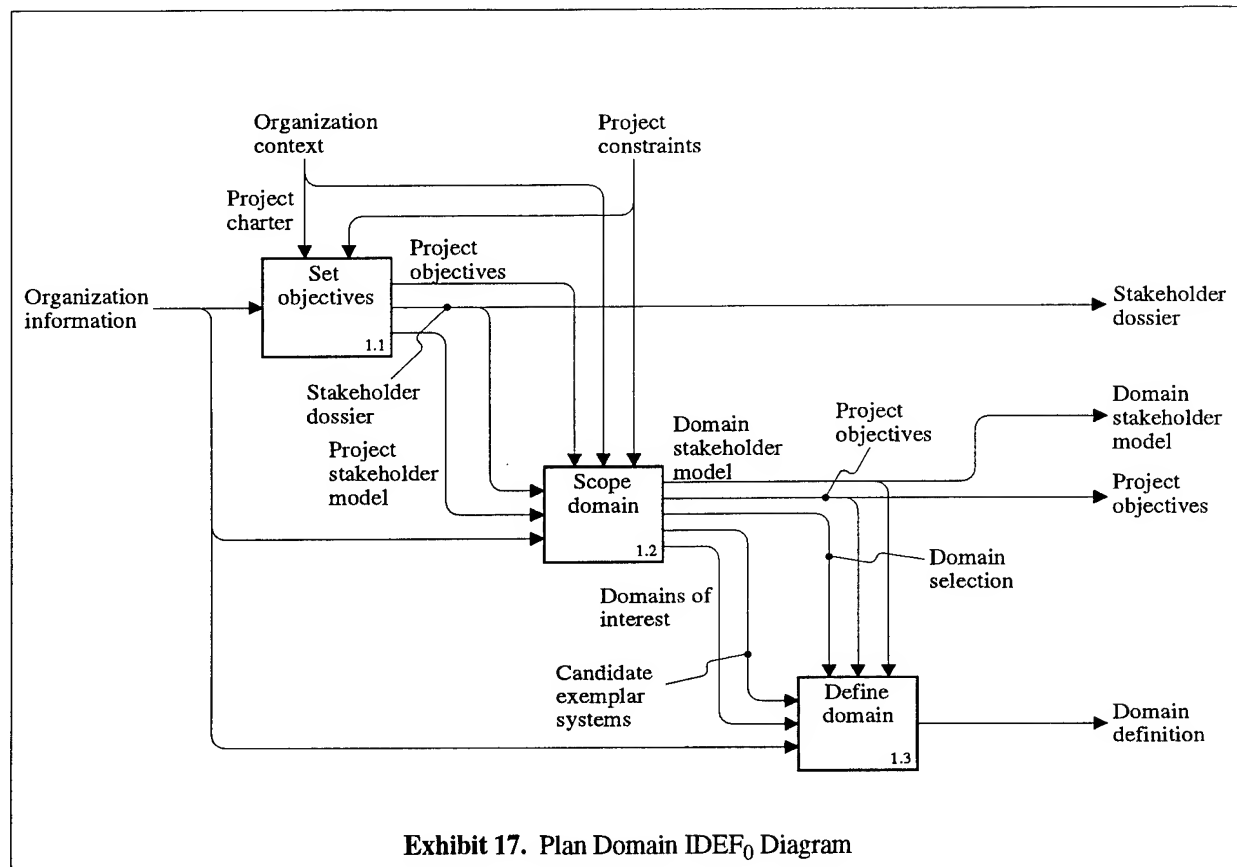
Exhibit 17. Plan Domain $IDEF_0$ Diagram

Domain Planning in ODM

One of the central aims of the overall *Plan Domain* phase in ODM is to ensure that a domain of the right size and characteristics is selected for the project, to ensure chances of success. The problem is getting people to see domain opportunities in between large and small scale concepts of domains. The idea of a large-scale domain (e.g., banking applications, flight simulator software) is intuitive and easy to grasp. The idea that "horizontal" components like window management routines might be used across these "application area domains" or "vertical domains" is also easy to see.

However, it is harder to "get" the idea that in designing assets for a "horizontal" domain we embed contextual assumptions about the vertical application areas in which it will be used (even if we are "building for reuse." Too often, engineers assume that when the problem scope drops below a certain line, the scoping problem goes away. This is not the case (e.g., specify a single "generic" component for manipulating text strings). Even to implement a "horizontal domain" such as window management systems requires making assumptions about the possible contexts of application. In implementing some sets of components for basic data structures, the variants are manifold and interacting, and soon introduce combinatorial explosion.

Similarly, we can imagine decomposing a large domain down into sub-domains, some of which might be still application area-specific, others more broadly applicable. It is harder to visualize each one of those components potentially being reused within other vertical contexts. Once this many-to-many picture is grasped, it is easier to see why domain engineering offers a significant methodological challenge. Where do you start? and how do you know you're finished? How general do you make the individual components?

61

The approach ODM takes to this problem is to treat the process of domain selection, scoping and modeling completely consistently regardless of whether the domain is initially thought of as "vertical" or "horizontal." Though these terms are used in the guidebook for explanatory purposes but ODM does not depend on any consistent understanding of the terms. They are mentioned primarily because you need to know how people are thinking of their domains in order to know how to challenge that thinking and offer other alternative viewpoints.

In the ODM process domains are partly discovered, partly invented/designed, and partly negotiated. These sorts of domains are at the same time the most strategically valuable for an organization, and at the same time may be hard for either business or technical people to see and recognize as an opportunity for domain engineering.

The ODM method directly reflects these principles

- System architectures may be important shared aspects of some application-area style domains, but not all of them and rarely for "horizontal" style domains. Methods that embed a required system architecture step, whether intentionally or not, embed a "weighting" towards certain styles and granularities of domain scope.

  ODM avoids embedding a system architecture step in the process. (The step called *Architect Asset Base* is a very different process, not trying to produce the same results.) This allows the method to be used without "hacking" for domains of widely varying granularity, and for domains where the assets to be produced are not software components at all.

- Most large-scale applications will consist of functions covering many domains. In a mature reuse-supportive environment, the application engineer would synthesize reusable assets drawn from many different domains and compose them into systems. For this to work, our methods need to help us discover relatively fine-grained, flexible and mutable domains.

- Understanding the boundaries of the domain therefore becomes almost as important (or perhaps more important) than having a detailed model of the functionality within the domain. This is because each boundary represents a link with another domain, either represented with its own model or there as a potential new area to be addressed. The domain assets will not exist as a resource in isolation.

The ODM domain planning process provides very detailed methods for getting engineers to be explicit about the "assumed range of applicability" for the domain. The ODM definition process also specifically addresses the characterization of the relationships of the domain of focus to other domains. This helps to identify potential for interoperability, adaptable configurations, variant architectural approaches, etc. It is therefore a method that is designed to scale up and forward to support the broader goals of an emerging components-based software industry.

Since ODM can be used in a consistent way for different domains, the possibility is heightened that assets from multiple domains engineered using the ODM method will be able to integrate well together. Of course, the full problem of semantic interoperability between separately developed domain resources will remain an active and challenging research area for the foreseeable future. But using a consistent method for the different efforts will be a good starting point.

Results

At the conclusion of the *Plan Domain* phase, a domain of focus has been selected and defined for the domain engineering project. The domain has been selected in alignment with explicitly identified stakeholders and objectives. Assumed objectives and domain scope from the PROJECT CHAR-

TER have been validated, and possible synergy with other objectives and stakeholder interests has been explored to maximize the potential payoff from the project.

Process

The name of this first phase of the ODM life cycle may suggest that it serves a conventional project planning function. Though these activities do need to be carried out in concert with the activities in this phase, the core ODM process model excludes most typical project planning and infrastructure planning activities. The process model and the sections that follow focus primarily on differentiating aspects of domain planning.

As shown in Exhibit 17, there are three sub-phases in the *Plan Domain* phase:

- In the first sub-phase, *Set Objectives*, domain engineers set the overall context for the project by establishing an explicit set of PROJECT OBJECTIVES in relation to candidate and selected project stakeholders, documented in a PROJECT STAKEHOLDER MODEL. This phase is controlled by the ORGANIZATION CONTEXT in which the project is initiated. Part of this context is a PROJECT CHARTER that establishes an overall mandate for the project; it may be formal or informal. The process does not assume that the PROJECT CHARTER constitutes final commitment to perform the project; articulating and obtaining key stakeholder commitment to the objectives is a critical element of this sub-phase.

- The primary output of the second sub-phase, *Scope Domain*, is the DOMAIN SELECTION. Also, in this sub-phase planners refine the PROJECT STAKEHOLDER MODEL into the DOMAIN STAKEHOLDER MODEL. Information about other domains relevant within the project context is captured in the DOMAINS OF INTEREST report.

- The primary purpose of the third sub-phase, *Define Domain*, is to formalize and validate the informal understanding of the domain resulting from the *Scope Domain* sub-phase. This process involves establishing domain boundaries through a combination of criteria and identified examples and counter-examples, situating the domain in terms of its historical context, structural interactions of domain functionality with other system elements, and conceptually related domains. These tasks and workproducts are a unique emphasis of ODM. Final decision to proceed to detailed domain modeling is not made until this definition is complete.

Including domain definition at the end of the planning phase is a distinctive feature of the ODM process model. It might appear to make more sense to worry about definition when you are ready to begin detailed modeling in the *Model Domain* phase. However, by performing *Define Domain* at the end of the *Plan Domain* phase, the entire phase produces more stable results. There may be some discontinuity in time and personnel between phases; with the current structure, the DOMAIN DEFINITION has a better chance of communicated to those who are going to go on to *Model Domain*. The definition process also helps reveal necessary points of communication with other projects and technology areas.

Guidelines

- Make objectives explicit. Teams attempting domain engineering for the first time may be tempted to perform domain selection with only informal PROJECT OBJECTIVES, skipping the *Set Objectives* sub-phase. It is better to document explicit PROJECT OBJECTIVES *before* selecting the domain of focus. The objectives will evolve. Even when a good-faith attempt is made to document PROJECT OBJECTIVES, some iteration back to the *Set Objectives* sub-phase will be inevitable. But the initial attempt will still be of value in clarifying hidden agendas and constraints on the project.

Skipping this first sub-phase may cause implicit objectives to surface when they are indirectly raised as issues as domain choices are investigated. If explicit objectives are lacking, there may be insufficient basis for selecting among candidate domains in the *Scope Domain* sub-phase.

- <u>Scoping may begin before objectives have been selected</u>. The *Scope Domain* sub-phase may be started before the *Set Objectives* sub-phase is completed. The risk is that domains that are not of strategic interest will be explored, since project stakeholders are not determined until after project objectives are selected. However, this is not necessarily a bad outcome if resource expenditure is reasonable, because it supports the goal of providing the organization a "domain-oriented" perspective on organizational and system boundaries.

- <u>Expect iteration in domain scoping and definition</u>. Domain scoping and definition may reveal stakeholders not anticipated in the initial project context (e.g., potential affiliations with other parts of the organization or other organizations.) These changes might necessitate revisiting the *Set Objectives* tasks.

- <u>Sequence when domain of focus is mandated</u>. Sometimes a domain of focus is mandated in the PROJECT CHARTER. At minimum, there will typically be a broad business area specified that indicates the organization entities considered within the project scope. Such a mandate on the domain of focus may appear to obviate the need for a formal domain selection process; it suggests that tasks up to and including domain selection are superfluous and can be skipped, and that domain engineering can begin with the *Define Domain* task.

We recommend following through the process as documented anyway, for several reasons. There may be hidden stakeholder issues that create strong elements of risk for the mandated domain. Characteristics of the mandated domain may prove incompatible with other explicit or implicit PROJECT CONSTRAINTS (such as the use of a particular technology). Or the domain may be scoped far too broadly for a viable project. There is still a need, therefore, to validate the domain selection against organization needs, to identify possible synergies and conflicts with stakeholders, and to scope the domain appropriately.

If the domain of focus is mandated in the PROJECT CHARTER, planning processes through the *Scope Domain* sub-phase can be performed as a review, validation and scoping of the selected domain, perhaps even in parallel with *Define Domain* activities. In addition, the mandated domain may be too large to address based on given PROJECT CONSTRAINTS. In this case, *Scope Domain* and *Define Domain* can be performed iteratively to progressively narrow the domain scope. However, this iteration cannot be done effectively if the initial context-setting steps are skipped.

Many of the guidelines above have a common theme: *don't skip steps*! The *Plan Domain* phase may seem to be asking you to assemble a set of information that you may think you already have in place. But skipping steps may lead to breakdowns farther along in the process, where difficulty in obtaining consensus will provide insight about missing or undocumented criteria for the domain of focus selection. At this point, "rolling back" may be costly or even infeasible. Or you may lose opportunities to find synergistic new stakeholders, project outcomes or candidate domains. Or there may be no particular breakdown, but in the end you will launch a domain modeling effort that will not "take" within the organization. Our goal is to create a domain model and asset base that live on as evolving resources, not an exercise. The steps as documented are necessary to *make it real*.

Example. As a real example, the Rolls-Royce trial experience cited in [Kell96] did a run-through of the ODM process in the domain of aero-engine systems, based almost entirely on working from the guidebook (version 1.0) directly. The project team chose to skip the first two sub-phases of the *Plan Domain* phase and begin with *Define Domain*. Their lessons

learned include two revealing insights.

— First, they found it hard to define the domain at the start. They attributed this to the lack of domain experience of the modeler. However, they also had no *strategic* picture of why they were defining the domain from a stakeholder perspective.

— When they reached the *Engineer Asset Base* phase, they found it difficult to get the domain experts to "commit to the future." There was insufficient certainty to motivate investing in the asset base within the domain.

Performing the first two sub-phases of *Plan Domain* would have increased the chances that these issues would be confronted at the outset. In this case, no great harm was done, because the trial project was exploratory. But even a research-oriented domain engineering project, if it is going to really enact the process, should enact it "for real," taking seriously the goal of establishing an ongoing asset base capability within the ORGANIZATION CONTEXT.

Example. As another example, another project with which we were involved made a similar decision to skip the first two sub-phases because "we already know our domain." In this case, the project never progressed much past the *Define Domain* sub-phase. Considerable effort was expended on iterating over issues that invariably traced back to unresolved stakeholder issues. The moral of this story is that even when the breakdowns happen soon in the process, they are not always recoverable. Dynamics can be created that affect the overall outcome of the project. In short: *don't skip steps*!

5.1 Set Objectives

In this first sub-phase of the *Plan Domain* phase, there will typically be some known stakeholders (primarily a sponsor) and a known objective. But ODM takes a somewhat pessimistic approach. For a domain engineering project to succeed there must be a "constellation" of multiple stakeholders that have agreed on some common objectives that are in alignment with their various interests. The primary purpose of the *Set Objectives* sub-phase is to explicitly document a viable set of PROJECT OBJECTIVES and characterize them with respect to the interests of an explicit set of these project stakeholders.
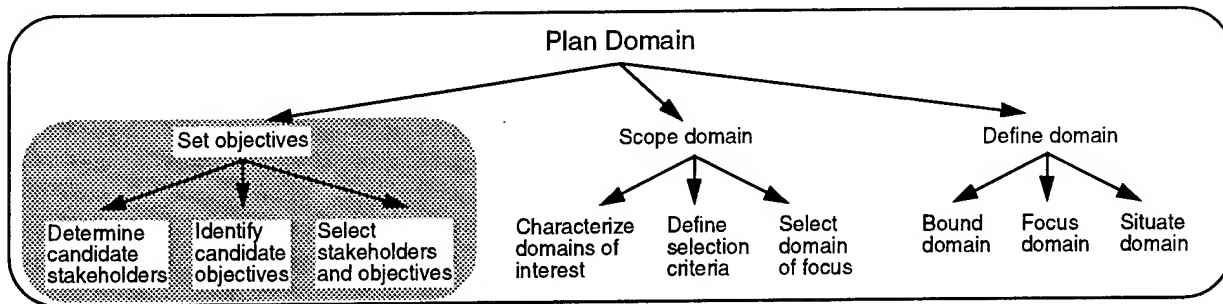
Exhibit 18. Set Objectives Process Tree

There are several challenges in establishing this strategic context for the project. There can be many motivating factors behind an organization's decision to explore domain engineering. Often the business context and rationale for a domain engineering project are not explicitly documented at the start of the project. There may be "hidden agendas" for various stakeholders, and conflicts that are not amenable to open discussion. It can be a challenge for the organization to identify objectives that are specific to reuse; this itself may involve some technology transfer to stakeholders in the organization.

Throughout the ODM process there is an assumption that the domain engineering team may cover a wide range of people in terms of their relative position in the stakeholder organizations and their relevant domain experience. Modelers could be insiders or outsiders (e.g., external consultants). They could be domain experts themselves or unfamiliar with the domain, e.g., recent college graduates who know advanced software engineering methods but have no basis in the applications within the line organization. They could also be more or less experienced in domain engineering skills and interactions. Each of these situations will present different challenges to the planning team.

Approach

A sustainable domain engineering project requires multiple stakeholders whose interests link together into a "reuse marketplace" configuration. This configuration can't be manipulated by management mandates; the stakeholders have to "come to the table" willingly. Typically, the configuration will also be something new for the ORGANIZATION CONTEXT. A new domain capability will be put in place; this inevitably will shift some stakeholder relations. Clearly, this is not just a matter of documenting a set of "given" project objectives. Setting publicly agreed upon objectives at the beginning of the project based on the interests of project stakeholders can help the project to focus on realistic objectives that will motivate stakeholders to buy into the project.

In addition to looking to multiple stakeholders, the process involves recognition that there are both enablers and barriers for various stakeholders with respect to a domain engineering initiative. As a simple example, domain engineering may involve codification of expert knowledge.

Exhibit 19. Set Objectives IDEF$_0$ Diagram
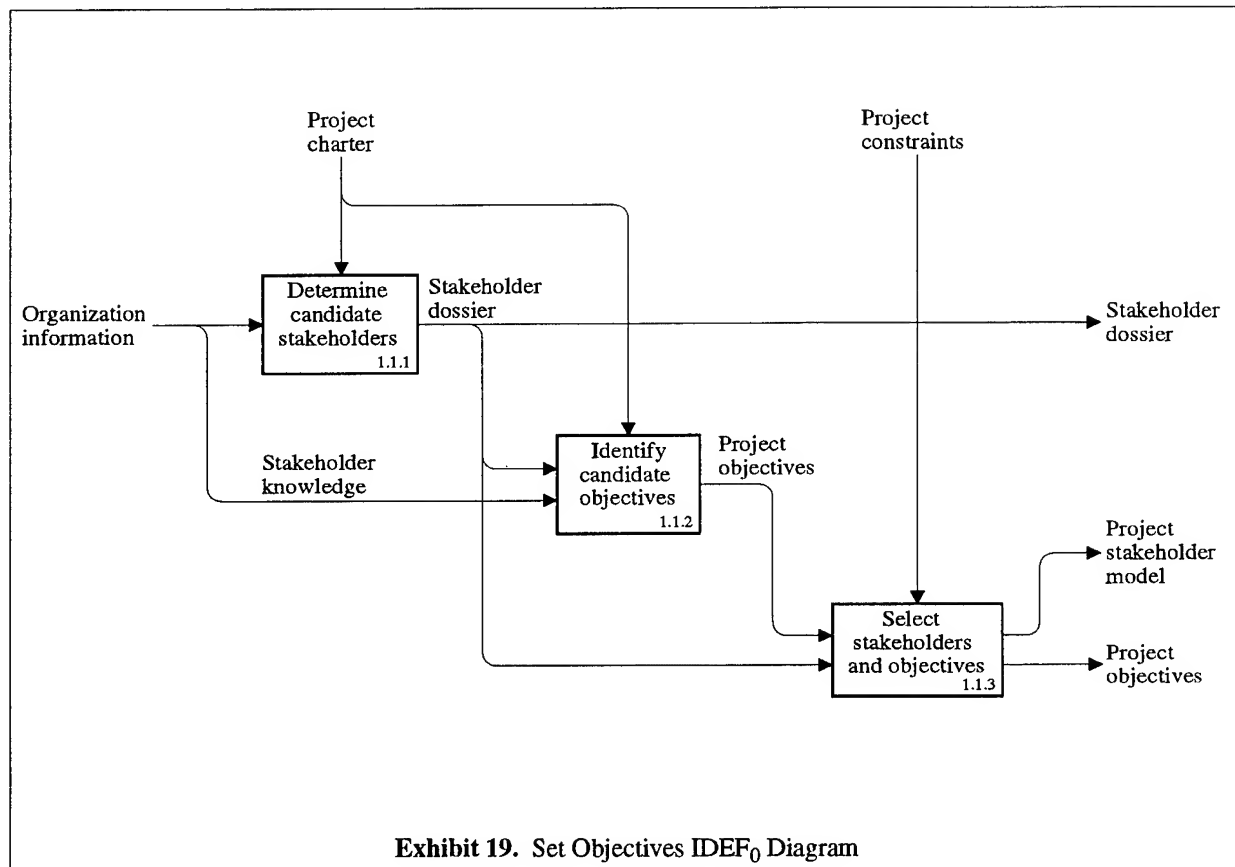
This could be viewed as a threat to the job security or "guru" status of the expert. It could be viewed as yet another "unfunded mandate" for more work passed on by upper management. Or it could be viewed as the person's chance to pass on their "legacy" to a new hire. Building the picture of stakeholders and objectives means painting in the "lights and the shadows," not just the conditions that appear to support the project.

The overall approach to the sub-phase thus involves building a "linkage" between stakeholders and objectives. Initial lists of both stakeholders and objectives can be derived from the strategic picture in the ORGANIZATION CONTEXT or "seeded" via reference to some standard stakeholder roles and typical reuse objectives to consider. The linkage between stakeholders and objectives is through the intermediary of *stakeholder interests*. These interests relate to overall strategic goals that are relevant to domain engineering objectives. Interests can be either "pro" or "con": that is, they can reflect reasons why the stakeholder would prefer, or would not prefer, a given outcome.

These various stakeholder relations, interests, and linkage to candidate objectives are explored through a *modeling* process that is a precursor to the formal modeling techniques and skills that will be required later in the domain engineering life cycle. (In addition to acting as technology transfer of the modeling skills, this "formality" is a deliberate "trick" to get stakeholders who come from a technical background to be willing to pay attention to the "soft stuff"!)

Results

The primary output of the *Set Objectives* sub-phase is an aligned and correlated set of PROJECT OBJECTIVES and stakeholders and their interests documented in the PROJECT STAKEHOLDER MODEL. The resulting PROJECT STAKEHOLDER MODEL provides an initial basis for setting con-

crete and achievable objectives for the reuse program as a whole; these in turn provide a basis for selecting and defining an appropriate domain of focus. The stakeholder model and associated assessment information are used and further refined throughout the domain engineering life cycle.

Process

As shown in Exhibit 19, the *Set Objectives* sub-phase consists of three tasks:

- The first task, *Determine Candidate Stakeholders*, results in a model of candidate stakeholder organizations and their interests relative to the project.

- In the second task, *Identify Candidate Objectives*, planners determine candidate objectives and map them against identified stakeholder interests.

- In the *Select Stakeholders and Objectives* task a set of objectives are converged upon that are internally consistent, satisfy key stakeholder interests, and are feasible given PROJECT CONSTRAINTS.

If there is insufficient incentive to go ahead with the project, having assessed the real level of stakeholder commitment and made a reasonable estimate of the resources needed, then the project may be terminated at this point. This will be a continuing theme throughout the process; such termination is not in and of itself a sign of a breakdown in the process. In fact, if conditions were not conducive to a successful effort, such termination is a sign that the process did its job.

Guidelines

- The *Determine Candidate Stakeholders* and *Identify Candidate Objectives* tasks can be performed sequentially or in parallel, iterating between them as additional stakeholders and objectives are identified.

- Since the *Select Stakeholders and Objectives* task involves trade-off analysis between stakeholder interests and objectives, both the *Determine Candidate Stakeholders* and *Identify Candidate Objectives* tasks must be completed before the *Select Stakeholders and Objectives* task begins.

- Various sequences are possible for these tasks. A *stakeholder-driven* approach works from the stakeholders and their interests, and attempts to trace these to relevant objectives. An *opportunity-driven* approach works from possible objectives or motivations for the domain engineering project and works "backward" to identify possible stakeholders in the situation for whom those objectives would be germane. In a situation with few stakeholders (e.g., internal to an organization with a few large divisions) the former approach may be simplest. In more of a "marketplace" ORGANIZATION CONTEXT, it may make sense to use the objectives to filter possible stakeholders.

  The task structure reflects a descriptive to prescriptive rhythm which echoes the overall structure of the ODM process model and recurs throughout various other sub-phases. The initial two tasks are *descriptive* in nature: they emphasize generation of exploratory and divergent views; they are followed by a *prescriptive* task emphasizing convergence on commitments and decisions (*Select Stakeholders and Objectives*). The core process puts stakeholders first in order to emphasize the approach that seems easiest to "jump start" the process, controls risks more than expands opportunities, and thus preserve focus.

5.1.1 Determine Candidate Stakeholders

In this first task of the *Plan Domain* phase, we are essentially identifying the "cast of characters" within the ORGANIZATION CONTEXT for the domain engineering project. For projects that involve entirely internal participants, this is a chance to reflect on and document the implicit contextual knowledge that often shapes strategic decisions. It is also a chance to identify the "hidden" stakeholders that may not otherwise be considered in planning, and to look outside the initial "frame" of project sponsorship to enlist support from other potential stakeholders. If the project involves the participation of external consultants or technology providers, this task offers an opportunity for context setting and orientation.
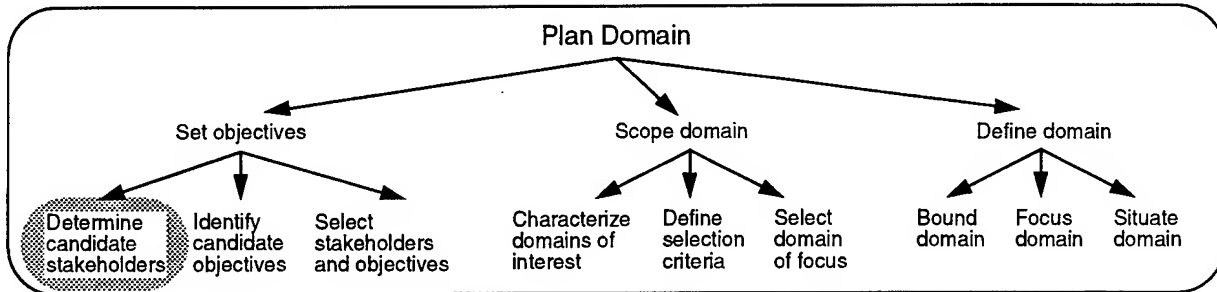
Exhibit 20. Determine Candidate Stakeholders Process Tree

The primary purpose of the *Determine Candidate Stakeholders* task is ensure that there is a coherent network for a sustainable reuse program. Most typically, we are seeking potential asset developers, maintainers, and utilizers in the case of an anticipated asset base. (In terms of the CFRP, this is a Create - Manage - Utilize or C-M-U process configuration [CFRP93a, CFRP93b].)

One challenge in this task is to assess these kinds of relationships within the organizations prior to domain selection. However, there are usually broad indicators that certain organizational divisions or entities are better positioned structurally to make the shift to a configuration supporting systematic reuse.

> *Example.* In a domain engineering initiative planned within a government procurement organization, some contractors are contracted to develop several variants of the same product within one contract "umbrella." This is a case where there could be incentive for reuse at the contractor side, or where contractual vehicles could be created to motivate this. A C-M-U style analysis could highlight this potential configuration as three Asset Utilizer groups within one organizational scope.

Another challenge in this task is the impact of individuals, with their values and beliefs, personal motivations and interests, and past experiences with reuse on the stakeholder picture. Decision-makers are not transparent actors on behalf of the business interests of the organization they represents. Stakeholder modeling must deal with both people and organizations as stakeholders in a strategic sense. This is particularly true with regard to an issue like obtaining sponsorship

Approach

The main approach to the *Determine Candidate Stakeholders* task is to analyze stakeholders to determine their interests, enablers, barriers, and relationships, based on overall information about the project's ORGANIZATION CONTEXT. Stakeholder information is placed in the STAKEHOLDER

DOSSIER. This information may be well-documented in the PROJECT CHARTER, or may be informal and implicit, requiring a fair amount of analysis and reflection to articulate.

A *stakeholder* with regard to a given issue, decision or initiative is someone who has a definable interest in the outcome of that issue, decision or initiative. This does not include supporters only; strategic opponents or competitors need to be considered as part of the stakeholder picture for a given situation. Stakeholders can be analyzed at the level of organizations, roles within organizations, or specific named people (especially in the project's organization). Stakeholders can have multiple *interests* in the issue at hand, including interests which may conflict. Stakeholders can also perform multiple roles.[1]

In domain engineering, analysis of stakeholders is essential because stakeholders are likely to span project, functional, departmental and organization boundaries. In the context of this task, *project stakeholders* are parties whose have an interest in the domain engineering project. This interest could be positive or negative; stakeholders include both parties that are advocates of reuse and those who are opposed to reuse. Project stakeholders could include including project sponsors, domain engineering technology providers and modelers themselves.

In this task, we deliberately "cast the net wide" to consider the broadest set of possible stakeholders for the project. *Candidate project stakeholders* include, in the broadest sense, all stakeholders who have an interest in the application areas within the organization's focus. Candidate stakeholders also include both people with a current interest in the organization's application areas and people who *may* have an interest in the future, based on projected changes that will occur as a result of the domain engineering project. They may include both people with a positive view of reuse, and those with a negative view of reuse, including skeptics within the technical groups, or proponents of other initiatives competing for the same sources of support as the domain engineering project. *Key stakeholders* are those whose buy-in and commitment is critical to the success of the project. Stakeholders found in the PROJECT CHARTER (including the funders) are usually key stakeholders.

The stakeholder picture for the project will be tracked throughout the domain engineering life cycle. At every point where a key scoping decision is made (such as selecting the domain or selecting the set of representative systems to study for the domain) the stakeholder picture changes. Not only the interests, but the set of stakeholders themselves may shift at these junctures. However, in this first task, as much as possible, we want to consider each of the different stakeholder roles that will be essential over the course of the project. Here is an overview of the most important stakeholder groups considered in the process:

- *Domain stakeholders* are those with an interest in the domain of focus that is selected for the project. Since projects will often include domain selection within their process, or will at least refine and downscope an initial domain scope suggested by the PROJECT CHARTER, selection and definition of the domain will create an overlapping but distinct set of domain stakeholders relative to the project stakeholders, those with strategic interests in domain engineering results focused in that domain.

  One important group of domain stakeholders are the *domain practitioners*. This term broadly defines anyone who works (or plays!) in a domain-specific setting. These might include software developers who build application in the domain, operators and end-users, managers who oversee applications development, marketers, retired experts, or anyone else who has

---

[1] Individual stakeholders are not equivalent to roles, however. The real person performing the role brings their unique perspective, which is captured to whatever degree appropriate in the stakeholder analysis. Stakeholder analysis, therefore, could consider issues like a stakeholder ambivalence, trying to perform multiple roles with conflicting interests in a given situation.

knowledge about the domain from practice.

- The *project team* (variously referred to at different points in the process as the *[domain] planning team*, the *[domain] modeling team*, and the *asset base engineering team* (or more simply, planners, modelers, and asset base engineers) are those who are involved in carrying out the specific activities of the process. It is an important element of the process to consider this group as a distinct stakeholder element. The background, broader organizational roles, and motivations of this team will dramatically affect how the process unfolds.

- *Informants* are, most generally, people and organizations that will become sources of information about the domain of focus. In the *Model Domain* phase of the domain engineering life cycle, which emphasizes descriptive modeling of domain capabilities rather than design choices about what assets to build, this is the primary role of domain practitioners external to the project team who are involved in direct contact with the team. (Ancillary roles could include managers who provide access to informants and make their time available.)

- Last but certainly not least, *customers* are those stakeholders who will be potential direct users (or indirect beneficiaries) of domain engineering results. This term is used most specifically in the *Engineer Asset Base* phase to refer to potential *asset base customers*, including the organizations that create a market for the asset base and the direct *asset utilizers* within those organizations. (Note that if domain engineering is performed within a software engineering organization, the organization's customers might not equate directly to the asset base customers; e.g., the latter might be internal to the organization.)

Example. In the Persona scenario, we have already described some of the key stakeholders in the business situation (or ORGANIZATION CONTEXT). Note that Maskara and Webonautics are two groups within one company, Persona. Clearly, these groups have different stakeholder interests that need to be considered, and these are not simple "data points" in a trade-off analysis. The groups have different strategic objectives and different relationships with mature vs. emerging technology "windows." There is even a different cultural ethos in the two groups, despite their being housed in one organization. The Persona organization as a whole needs to be considered as a distinct stakeholder from either individual group. The president of Persona, Chris Connell, is also a distinct stakeholder from Persona, not merely a "spokesperson" for the organization. He believes the cultural "rift" between the two groups is an issue that needs to be addressed; this is to some extent an atypical "inward" concern for a company leader, and might not be an issue for a different individual in the same role.

Thus stakeholders (including "customers") do not correspond one to one with organizational boundaries. Internal groups within organizations can be distinct customers with respect to domain engineering. This extends not only to strategic interests but to "developers' culture" as well.

Note also that the two groups have overlapping but diverse technology areas of concern. This creates is a strategic situation allowing for a domain engineering opportunity in part because these two groups are in the same organization; if they were two separate companies there might be no clear stakeholder to "own" the potential asset base (e.g., the two companies alone wouldn't be enough of a market to motivate a separate components provider). On the other hand, if there weren't significant variability in the technical requirements of the two groups, then the groups could simply share a common product and would not need to consider more systematic approaches to reuse like domain engineering.

71

Workproducts

■ STAKEHOLDER DOSSIER

The STAKEHOLDER DOSSIER contains a list of candidate project stakeholders, partitioned into key and other stakeholders. The dossier also contains information about stakeholder roles, relationships, interests, enablers and barriers, and assessments of attitudes towards reuse and reuse practices. Exhibit C-1, PROJECT STAKEHOLDERS/ROLES MATRIX, contains a worksheet template that can be used check the list of stakeholders for completeness with respect to the fundamental domain engineering roles. Exhibit C-2, PROJECT STAKEHOLDERS INTERESTS TABLE, contains a worksheet template that can be used for identifying stakeholder interests, and enablers and barriers to stakeholder buy-in based on each of these interests. The STAKEHOLDER DOSSIER will be used in coordination with candidate objectives to determine the final PROJECT OBJECTIVES in the *Select Stakeholders and Objectives* task.

When to Start

- Initiative launched. An initiative has been launched for a domain engineering project.

- Project mandate has been documented. The mandate for the project has been documented in the PROJECT CHARTER.

Inputs

- ORGANIZATION INFORMATION. Information about stakeholder organizations, used to develop the model of stakeholder roles, interests, etc. General sources of ORGANIZATION INFORMATION may include: organization charts, business plans, marketing literature, annual reports, and informal knowledge of project members. Other valuable sources of information are results of reuse-specific and broader organization assessments.

  Assessments required for planning domain engineering may overlap to some extent with wider organization assessment initiatives including:

  — Software engineering maturity assessments such as those performed using the SEI's Capability Maturity Model (CMM)

  — Formal reuse assessment instruments, such as the Software Productivity Consortium's Reuse Capability Model (RCM) [SPC93a], or the STARS Reuse Strategy Model [RSM93a]

  — Informal surveys,

  — Software productivity assessments, or

  — Organization assessments for strategic planning, quality programs, or line-of-business initiatives.

  Assessments can be useful in establishing an accurate picture of the current state of an organization. However, general assessment of the entire organization is too broad a task for the purpose of setting PROJECT OBJECTIVES. Utilize general assessment information that is available, but keep the project focus in mind in initiating other assessment activities. Remember that the ORGANIZATION CONTEXT does not necessarily map to a single organization: it may be a smaller division, or a consortium or industry group that spans many organizations. Any organizational assessment may therefore cover too much and/or too little ground—and will

generally have done assessment with regard to very different objectives.

Controls

- PROJECT CHARTER. The official mandate to perform the domain engineering project. There may be a well-defined PROJECT CHARTER or only a vaguely understood mandate for the project. Ideally the charter would describe the ORGANIZATION CONTEXT of the project, including whether the initiative is part of a more comprehensive reuse program. The PROJECT CHARTER often names key project stakeholders. It also defines the scope of interest for the project in terms of organization structure (who can be tapped, who must be included, who must be left alone to get "real work" done).

Note that PROJECT CONSTRAINTS are *not* a control to this task. Stakeholder interests should not be screened based on project resources at this point. Here the intent is to identify as many reinforcing and competing interests as possible. Screening of stakeholder interests is done in the *Select Stakeholders and Objectives* task. (Of course, the task itself has constrained resources for performance. This is assumed for all tasks in the process model and not deemed necessary to document.)

Activities

The *Determine Candidate Stakeholders* task involves identifying stakeholders and analyzing the stakeholders and their organizations. Some general techniques for this identification and analysis are presented here. Stakeholder analysis techniques from other disciplines can be applied in these activities, as is discussed in the Stakeholder Analysis supporting method area in Section 8.1.

➤ Identify candidate project stakeholders

Candidate project stakeholders are groups and individuals who have an interest in the application areas within the organization's focus. There will be a unique configuration of stakeholders for each project in which domain engineering is initiated. Candidate project stakeholders may include managers, software engineers, test developers, field technicians, technical support staff, documentation and training staff, end users, value-added resellers, marketers, and competing organizations. Stakeholders can play multiple roles. If technology selections have been made, technology developers should also be considered stakeholders.

To ensure that no candidates are overlooked, use two or more of the following methods to identify candidate project stakeholders:

- Look in the PROJECT CHARTER for stakeholder names and roles.

- Brainstorm candidate stakeholders based on ORGANIZATION INFORMATION elicited from project members' informal knowledge. Then determine the roles held by these stakeholders.

- Talk to known stakeholders within the project's organization and generate names of other candidate stakeholders based on ORGANIZATION INFORMATION held by these known stakeholders.

  Example. Candidate project stakeholders for a manufacturer of software-intensive peripheral equipment such as laser printers might include:

  — Software developers,

  — Integration testers,

— Field technicians from the producer organization,

— Customer system managers who install new software upgrades,

— Third-party software vendors who write driver software to make use of the peripheral's software, and

— End users of the equipment.

➤ **Identify stakeholder roles and candidates for each role**

Ask questions based on critical roles to identify stakeholders who fill these roles. Exhibit 21 shows a starter list of questions, based on roles that are often critical to a domain engineering project.

Who are core or key customers of the domain engineering project?

Who is funding the project?

Who defines success for the project? (e.g., funder, the funder's customers?)

Who will be performing the project?

Who are experts on application areas within the organization's focus?

Who are potential customers for assets to be developed by the project?

Who is providing technology for the project?

Who might be interested in seeing the project *not* succeed?

Exhibit 21. Starter List of Stakeholder Roles

Other questions to ask include the following:

- Based on organization relations (e.g., external market relations, internal customer-supplier relationships), which organizations are potential asset developers, asset base managers, utilizers of assets? Who are stakeholders within these organizations?

- Look at previous reuse efforts within the project's organizations. Results of previous domain engineering projects, both formal and informal, need to be considered early in the *Determine Candidate Stakeholders* task. These results can lead to the identification of stakeholders who were involved in the previous projects.

Following is an example of stakeholder roles:

Example. In the Persona scenario, other stakeholders include some large-scale customers with which Persona has or is attempting to establish relationships. While Persona has many clients these are all clients who are themselves software development organizations or related clients. These include the following:

— Air Force Distributed Simulation Office (AFDSO), an Air Force Branch trying to bring some domain engineering based order to the chaos of one-off war gaming and simulation systems — their first order of business is to establish shareable structured information views of some basic war game elements such as aircraft, satellites and radar installations.

— Medical Monitors, Inc. (MMI), a medical systems company (recently acquired by a division of a larger electronics products company) which builds safety and time critical mon-

itory systems for ER and IC wards. They are stressing the capability to monitor entire wards from a variety of stations and selectively monitor patients as well. Their systems require several views of various sorts concurrently. They use special purpose hardware and software for the actual data collection, but the viewing stations (with various analysis) will be on workstations. MMI has been a regular customer of PTI for several years; their current viewing software was done as a special one-off project by PTI, and (so far) has been maintained to meet their needs.

— PCGate Legal Services, a legal services company specializing in patent law with an interest in using real-time courtroom interactive fact and law software called PCLaw. This software would provide several things for clients (primarily lawyers), including a fact and law "tree"/taxonomy (FLT) specific to the case at hand, real-time translations of court transcripts and placement of result into the context of FLT, concurrent use by several attorneys (with messaging, etc.) including updates and queries. They wish to utilize laptop hardware and software.

— JSoft, a legal services software company partnering with PCGate to develop PCLaw. They have specialized in various legal software products targeting PCs.

— Finally, in order to ensure a viable proposal, Francesca Morrow decides to bring in an external consultant from ODM Inc. to aid in initial project planning.

➤ Characterize stakeholder organization(s)

In order to determine stakeholder interests, enablers and barriers, it is often useful to characterize the stakeholders organizations. Criteria that can be used to include the following:

- *Sector orientation.* Following are some examples of sector orientation:

  — government organization,

  — government contractor,

  — commercial software developer,

  — educational or research institution,

  — non-profit organizations (NGOs, etc.)

  Sector orientation strongly influences stakeholder interests relevant to the domain engineering project. For example, time to market may be a significant motivator for a commercial product developer, whereas reducing maintenance costs may be the major cost driver for a government maintenance organization focused on post-deployment system support. For a research or educational institution, obtaining breakthrough research results will be of high priority. Sector orientation can raise other potential issues, such as whether data that belongs to the organization is classified, sensitive, or public.

- *Business model.* Closely related to sector orientation is overall business model, including the following:

  — product-based,

  — contract-based,

  — research-based, or

— service-based.

Example. In the Persona scenario, the company as a whole is in the commercial software market. They do have a legacy of some government customers and some government contractor customers, but this is shifting. Their strong commercial orientation is in transition or flux around different business models. They are an example of a commercial contract-based company in the main; their services have generally been one-time engineering services to build specialized interfaces; their product lines are still emerging. To some extent, they are also an example of a "knowledge-based" company, in that their depth expertise in UI development is one of their primary competitive differentiators. This means expertise of particular staff is crucial to their economic growth and competitiveness—a potential point of vulnerability.

- *Life cycle phases supported by the organization.* Following are some example life cycle phases:

  — development,

  — prototyping,

  — maintenance, or

  — IV&V.

  Example. Persona has typically been involved in the prototyping and development cycle with their best customers. When they have caught the maintenance "tail" they have generally not fared well, in part because, as UI specialists, their portion of delivered systems have numerous interactions with code controlled by other parties. Their own product lines are still emerging, so they have not yet established mature full life cycle support in house for commercial products offered. This implies that reuse benefits such as ability to rapidly prototype new applications with impressive feature differentiation might be of more interest than a very controlled "product line" approach. (Compare this to, say, a large company needing interfaces to a large number of internal applications and desiring the greatest possible degree of uniformity across the different interfaces.)

- *Role and use of software engineering within the organization.* While related to the sector typology mentioned above, this aspect is still somewhat independent. Some example roles for software include the following:

  — The organization produces products with significant software components (e.g., electronics firms with embedded software in hardware products),

  — The organization produces primarily software products (e.g., a software tools vendor),

  — The organization offers services with an intensive software components (e.g., telecommunications firms, banks, insurance companies), or

  — The organization (typically large-scale) organization uses software to manage its own internal operations (e.g., factory automation, inventory control, MIS departments).

These aspects are somewhat independent; for example, government organizations could perform multiple roles. For many large enterprises (e.g., commercial banks, airline companies) the volume of software developed to support internal business operations will justify applying a domain engineering approach. In organizations involved with software engineering products or contracts, there is traditionally a split between the engineering and internal software groups. Opportunities for domain engineering may occur in both areas. Viewing the role

of software within the total organization can facilitate restructuring the organization along the lines of knowledge domains.

> Example. Persona is definitely in the software engineering business, as opposed to an enterprise needing better reuse within internal operations software. This means the pay-off for them will lie in domain engineering efforts applied to the applications they sell to their customers. However, as they are also service-oriented, and sell in a niche (user interfaces) that traditionally has a "component" or enabling technology flavor, another option to consider is that they could broker assets (or even asset bases) to customers or technology partners. Whether this would be strategic for them is, of course, a more complex question.

➤ Match stakeholders to roles

Working from the previously developed lists of stakeholders and roles, and the initial characterization of the stakeholder organizations, check the list of stakeholders for completeness with respect to the fundamental domain engineering roles. The purpose of this activity is to ensure that all roles held by each stakeholder are identified. Exhibit C-1, PROJECT STAKEHOLDERS/ROLES MATRIX, contains a worksheet template that can be used for this activity.

Determine which stakeholders are key stakeholders, whose buy-in is critical to the project. Stakeholders found in the PROJECT CHARTER (including the funders) are usually key stakeholders. Extend the roles to include other roles that are relevant to the situation. Some stakeholders may have multiple roles. Once both stakeholders and roles have been identified, look for more roles held by each stakeholder. For each stakeholder, determine whether the stakeholder can fill each identified role. Validate that all critical roles are accounted for, and that all key stakeholders have at least one defined role.

> *Example.* While we will present only excerpts from most workproducts for the example domain, the example PROJECT STAKEHOLDERS/ROLES MATRIX is useful to include in its entirety because it provides a convenient "snapshot" of the various stakeholders involved. It is depicted in Exhibit 22.

> Key stakeholders include individuals (Connell, Morrow, Perricone) and organizational enti-·ties (Persona, Maskara, Webonautics). Note the "web of interests" that make each stakeholder key in this situation. A viable project could be defined for either Webonautics or Maskara independently; but this would not address Connell's interest in fostering more communication between the groups. A viable joint project could be defined that did not include Jack Perricone's buy-in; but then Connell's concern about an overly insular reuse pilot project would not be addressed. Morrow, as the reuse advocate in the situation, is the "prime mover" behind the project and is therefore also an essential player.

> Note also that Persona, Maskara and Webonautics occur as key *and distinct* stakeholders. In other words, the organizational scope of different stakeholders can overlap. This can be a difficult concept to grasp. The strategic intent of a given stakeholder is a result of the potential economies within that stakeholder's scope of responsibility, authority and potential reward (where scope could involve organizational breadth or time horizon).

Stakeholders		Roles				
		Sponsor	DE	Asset Utilizers	End User	ODM Facilitator
Key Stakeholders	C. Connell	X				
	F. Morrow	X	X			
	Webonautics		X			
	Maskara		X			
	Persona		X	X		
	J. Perricone	X				
Other Stakeholders	JSoft			X		
	MMI			X		
	AFDSO				X	
	PCGate				X	
	ODM inc.		X			X

Exhibit 22. *Example:* Project Stakeholders/Roles Matrix

To be thorough, a separate worksheet could be created for the current roles within the organization and for the proposed new roles given the introduction of a domain engineering initiative. Since specific objectives and domain focus have not been made, there may be several alternative scenarios for the proposed new stakeholder "configuration." But there are also many constraints placed on what scenarios are feasible, given the overall business models, etc. that are in place.

Example. The worksheet example above is based on a scenario that assumes the initial domain engineering effort will be carried out by a team made up of F. Morrow herself, along with selected personnel from the Webonautics and Maskara groups. Morrow will also bring in the ODM Inc. consultant who will act as an ODM facilitator. Persona as a company will perform the domain modeling through the efforts of the domain modeling team already listed, as well as other personnel who will join the project as appropriate; also, Persona will make use of the reusable assets developed to provide services to their present customers.

JSoft, as an implementor of systems for legal services, and MMI, in a similar position for medical records, are also potential utilizers of the assets in the asset base. Although it would
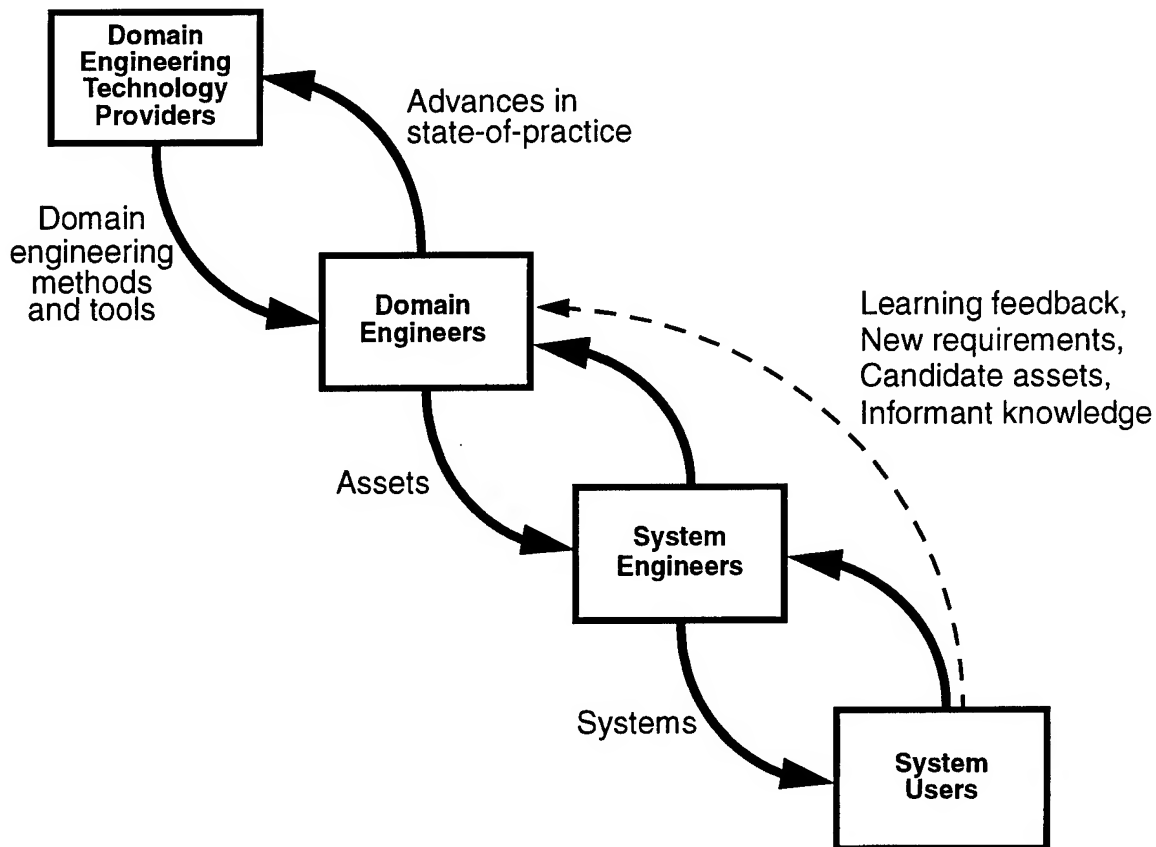
Exhibit 23. Stakeholder Relations Diagram

be useful if PTI could sell these assets to other companies, the fact that MMI and JSoft are listed as non-key stakeholders means that the success of the project does not hinge on having MMI and JSoft satisfied with the outcome. AFDSO and PCGate are known end users, not likely to play any role that produces software; hence they play only the End User role.

➤ Determine stakeholder relationships

Determine relationships among stakeholders. These can be done by tracing from the documented stakeholders' roles, or done separately and used to cross-check the roles. This is needed input to the final *Select Stakeholders and Objectives* task, because stakeholder relationships affect who's "in" and who's "out," irrespective of the roles each stakeholder plays individually.

Relationships can be shown graphically or in a matrix, as suggested by the typical relationships in a basic reuse/software "value chain" depicted in the graphical representation of Exhibit 23. As with stakeholder roles, you can make separate pictures to represent current and anticipated new configurations of relationships among stakeholders, based on the outcome of the domain engineering project. The conventions of the diagrams are simple; moving down and to the right means moving down in the "software value chain" towards the end-user. Entities drawn in the same diagonal (moving up and to the right) are at approximately equal positions in the value chain, and will tend to consumer and/or produce from common entities. Upward-flowing arrows generally represent learning and feedback and downward-flowing arrows, reusable assets or other enabling technology.
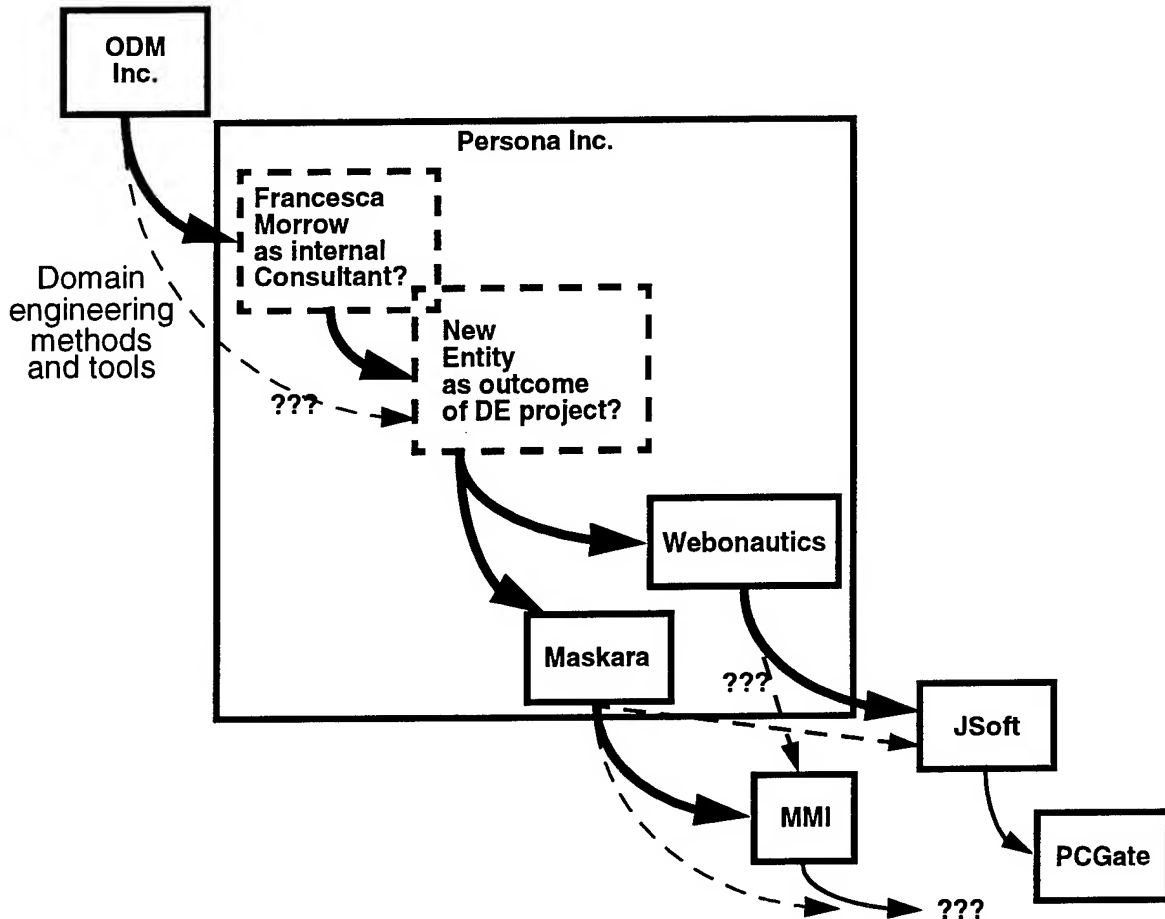
Exhibit 24. Example Scenario Stakeholder Relations Diagram

Example. In Exhibit 24 we see an example of a stakeholder relations diagram for the Persona scenario. (This is a fragment only; for example, no upward arrows are depicted to simplify the picture.) Here enclosed boxes suggest organizational "inclusion" and dotted boxes represent proposed future entities.

Even in this simple example we can see how the notion of the "value chain" can lead to complex stakeholder issues. This diagram suggests a proposed scenario where a new group will be formed to produce assets within Persona. This group will address some common requirements of both the Maskara and Webonautics groups and thus each of these groups will play an asset utilizer role. Presumably these groups would then use the assets to produce systems for JSoft and MMI more cost-effectively and competitively. The question marks between the Maskara/Webonautics and the MMI/JSoft "tiers" suggest an open issue: will both groups market to both customers? Or would an asset manager/broker role be appropriate here?

Higher up the "value chain" we see ODM Inc. as technology provider. But what will Francesca Morrow's ongoing role be? An internal consultant to the domain engineering team? A full-fledged member of the team?

➤ Cross-validate stakeholder roles and relations

Example. When we compare the data in Exhibit 22 and Exhibit 24 some inconsistencies emerge. These turn out to be valuable aids to identifying some of the strategic decisions that must be made.

It is clear that the picture of where the new asset creation group would "live" within the organization is different in the two workproducts. The roles view assumed that Webonautics and Maskara would be play asset creator roles, while Persona as a whole (meaning, in this case, a new entity within Persona) would play the asset broker or utilizer role. The relations view implies just the opposite; the new organization will play asset creator to both Webonautics and Maskara as asset utilizers. One possible reason for the confusion: the new group will presumably be staffed from Maskara and Webonautics personnel; but this does not determine whether it is a strategically new organizational entity or not.

This is an example of working representations in parallel as a validation technique. We have now identified a few alternative scenarios for domain engineering outcomes. In this case we adjust the stakeholder roles view to reflect the relations view. The excerpt in Exhibit 25 shows a revised PROJECT STAKEHOLDERS/ROLES MATRIX for the example. Note that in addition to shifting Webonautics and Maskara's roles to the asset utilizer roles, we have effectively "split" the single utilizer's column into internal and external utilizers, which reflects a bit better the value chain relations. (In general, the columns will need to be tailored to each ORGANIZATION CONTEXT, which will have its own unique value chain.)

Stakeholders		Roles		
		DE	Asset Utilizers-Internal	Asset Utilizers-External
Key Stakeholders	Webonautics		X-3	
	Maskara		X-3	
	Persona DE Team	X-2		
Other Stakeholders	JSoft			X-4
	MMI			X-4

Exhibit 25. *Example Excerpt:* Revised Project Stakeholders/Roles Matrix

➤ Characterize reuse practice of stakeholders

Some stakeholders in the ORGANIZATION CONTEXT will have been identified with the potential roles of asset creators, managers and/or utilizers. This means the success of the initiative may depend on a transition of these groups to more systematic reuse-based practice. In order to evaluate the feasibility of this goal, it is helpful to characterize current reuse practice in these organiza-

tions. This assessment should include determining the types of reuse (informal or formal) currently being practiced and receptiveness to reuse.

Survey results of previous reuse efforts, including domain engineering projects, within stakeholder organizations. Successful efforts should have produced data useful to the new domain engineering project. War stories about unsuccessful efforts may cause stakeholders to be extremely resistant to new attempts to introduce domain engineering practices. In many cases, previous efforts will have created vested stakeholder interests, and will have a major impact on the reception of new reuse efforts.

Assess the organization communication infrastructure, including internal electronic mail systems, libraries, corporate communication, training departments, and education departments. How much are these resources used for informal or *ad hoc* reuse? These resources will be used during domain modeling as a way to broadcast requests for domain information. Later, once an asset base is established, these same resources will be instrumental in technology transfer. Some useful questions to ask when assessing the organization communication infrastructure are:

— How are resources shared, if at all?

— What informal repositories are used?

— How do people know where to find things?

— Who are the veterans, the experts in particular subject areas that serve as informal "knowledge bases" for the developers?

Assess receptiveness of each stakeholder organization to reuse in terms of these key aspects:

— *Concepts*. The exposure of stakeholders to reuse concepts and methods.

— *Capabilities*. Level of maturity with regard to reuse technology and practices.

— *Beliefs*. Prevalent beliefs about reuse held by the group. For example, a group developing real-time applications may believe that attempts to reuse software will always result in inefficient code. This belief must be identified and understood by planners in order to arrive at objectives and measurable success criteria that will address these concerns.

These aspects are fairly independent. An organization may perform at a high level of reuse practice, yet be relatively unaware of formal reuse processes. This would be typical of a vertically focused applications shop with low turnover. Other groups may be aware of reuse terminology and concepts, yet be far from practicing reuse themselves. (This, sadly, is true of many of the contractor organizations that preach reuse via contract research and development but have made little progress in transitioning it into practice.)

➤ Characterize stakeholder interests

For all key stakeholders and a representative sampling of other stakeholders, identify and characterize their *interests* as they relate to the project. Interests may be expressed directly by stakeholders or inferred by planners. Knowledge of these interests assists planners in determining objectives and, later, a domain of focus. Tying objectives and domain of focus to stakeholder interests will support gaining buy-in and commitment from these stakeholders. Stakeholder commitment is essential for domain data acquisition and validation, and ensures use of the assets developed.

Stakeholders' interests may be motivated by many things. Where possible, look for interests that do not merely restate strategic issues captured by the current and potential roles and relations. For example, an individual stakeholder's interest in reuse is often connected with core professional or aesthetic values.

Each interest should be analyzed for the following:

- Enablers: What aspects of the project might promote the stakeholder's buy-in with respect to this particular interest?

- Barriers: What aspects of the project might create barriers that make the stakeholder's buy-in less likely?

    It is important to understand barriers to a reuse-based approach particular to the business environment in which the reuse program is initiated. Do not dismiss perceived barriers as mere resistance to new technology on the part of stakeholders. There will often be sound technical reasons for skepticism that should be respected.

Exhibit C-2, PROJECT STAKEHOLDERS INTERESTS TABLE, contains a worksheet template that can be used for identifying stakeholder interests, and enablers and barriers to stakeholder buy-in based on each of these interests. This takes the stakeholder analysis down to a finer level of detail, and allows for factoring in some qualitative data that reflects cultural attitudes, previous experience with reuse, etc. The stakeholder organization characterizations performed in earlier activities are important input for this worksheet.

Example. Exhibit 26 contains an excerpt from the PROJECT STAKEHOLDERS INTERESTS TABLE for the Persona scenario. This reveals that F. Morrow has an interest in enhancing Persona's overall technical reputation. Note that this is not a *personal* interest (as would be, for example, an interest in promotion within the firm) but it is a very different viewpoint about the critical issues for the company than C. Connell's interest in fostering greater collaboration between groups. Webonautics and Maskara both have a healthy disdain for management's 'flavor of the week' syndrome, and if domain engineering is presented in the wrong light this could be a strong barrier to acceptance. But their attitudes about the right methods for getting the "real work" done are somewhat divergent. Jack Perricone has a particular interest in partnering with MMI to land a big contract. This means a domain engineering effort focused on a vertical niche (particularly medical domains) would be a strong enabler for him. A solution bound to a single platform would create a barrier to his buy-in.

Note that neither the interests, nor the enablers and barriers, are yet dealing with specific domain engineering project objectives. These are the focus of the next task, *Determine Candidate Objectives.*

➤ Identify polarities among stakeholder interests

This activity is a suggestion only for one approach to eliciting and/or evaluating interests. In multiple stakeholder/multiple agenda situations, there is a common tendency for positions and beliefs to *polarize* in certain fixed ways. Rather than look for direct conflict in stakeholder relationships (e.g., competitive vs. cooperative relationships) or looking for agreement/disagreement about individual interests, it can be helpful to identify interests that seem to form polarities or dichotomies. These may concern technical issues (which architectural platform to adopt, methodologies) or business scenarios.

Example. In the Persona scenario, several key polarities emerge, including:

- The tension between focusing on the UI specialty versus picking vertical niche "beach-

head" markets as overall company strategy;

— The Web vs. MOTIF/OpenDoc/CORBA technology split; and

— Ongoing issues about business structure: product, selling turnkey systems, or ongoing consulting services (more a spectrum than a dichotomy).

Stakeholders Interests			Enablers	Barriers
Key Stakeholders	C. Connell	Greater collaboration between Webonautics and MasKara	Production of single library of interest to both	Separate projects
	F. Morrow	Give Persona reputation that attracts good technical people	Enter future-oriented technically viable area	Follow a fleeting fad
	Webonautics	'Real work' as opposed to management's 'flavor of the week'	Use of established methods (OO)	Overemphasis on non-technical reuse issues
Key Stakeholders	MasKara	Reduce production costs to be more competitive on government contracts	Systematic reuse	
		'Real work' as opposed to management's 'flavor of the week'	Use of established methods (Ada generics)	Overemphasis on non-technical reuse issues
	J. Perricone	Partner with MMI to get big CDC contract	PC look and feel compatibility	Solution bound to a single platform
			Vertical niche - medical	
Other Stakeholders	MMI	Get contract with CDC	PC-compatible (most popular system at CDC)	Insufficient attention to security

Exhibit 26. *Example Excerpt:* Project Stakeholders Interests Table

Example. The dichotomy between a UI and a vertical market niche will have considerable impact on domain selection. This polarity concerns what kinds of categorizations are potentially viewed as domains within the organization, and how they will be prioritized. The Web/CORBA polarity appears to be a technical issue, but in fact embeds a number of related cultural issues that relate to attitudes about software engineering practice and hence receptivity to reuse ideas within the respective groups. The business structure decision between a product, custom application and service orientation for the company in the future will have considerable impact on questions like how repeatable a domain engineering process would need to be within the firm, and who would own an asset base if developed.

Identifying these polarities is important in understanding how various technical objectives will capture or lose different stakeholders in the organization. Each polarity is a trap: choose a side and you have served to reinforce an organizational barrier; attempt a project scope to unify approaches viewed in this dichotomized form and you may encounter intense resistance. At least the analysis helps us know what we're up against!

When to Stop

- <u>Key stakeholders have been identified</u>. All stakeholders whose interests could determine success criteria for the project have been included in the STAKEHOLDER DOSSIER.

- <u>Additional stakeholders have been identified</u>. Additional stakeholders have been identified and have led to the discovery of opportunities for leverage and/or synergy and possible risk factors.

- <u>Key stakeholder relationships have been identified</u>. Potential conflict, trade-off, or dependency relationships among key stakeholder interests have been recorded.

  It is *not* necessary that identified conflicts be resolved. Since domain engineering often involves multiple stakeholders within or across organizations, conflicts and trade-offs may occur. The representation used for the STAKEHOLDER DOSSIER must be able to accommodate conflicting stakeholder interests. The process involves deferring attempts to resolve conflicts immediately.

Guidelines

- <u>Use collaborative self-assessment techniques</u>. There are a number of assessment instruments available that utilize an external auditing style of information gathering. Results of such assessments can be useful inputs to the *Determine Candidate Stakeholders* task. If assessment activities are undertaken as part of the domain engineering project, it is best to use a collaborative self-assessment style of information gathering.

- <u>Outside facilitation can be helpful</u>. Although no extensive external assessment is called for here, some involvement of outsiders can help at the initial stages. This external participation can provide an impartial "observer" role in stakeholder discussions, providing an element of "insider-outsider" dialogue (e.g., getting people to articulate what they "know" that turns out to be not as shared an understanding as was thought). This can have useful payoff throughout the ensuing processes.

- <u>Check for planners' bias</u>. In this task informal knowledge of project members should be used ·cautiously, particularly when the project team has a different background and culture from prospective customer groups (e.g., when a research and development-oriented group will be doing modeling to create assets for an application engineering group.)

- <u>Model diversity as well as consensus</u>. If there are multiple project customers, document interests unique to a customer as well as common interests. Also document apparent conflicts among stakeholder interests.

- <u>Consider competitive interests</u>. Stakeholders may have competitive relationships, especially in multi-organization project contexts. These competitive relationships must be identified in order to develop achievable project objectives. For example, for an industry consortium, an open system standard may be a feasible outcome of domain engineering, whereas a shared component base may be infeasible because of competitor company interests.

- <u>Consider interests that may change the business model</u>. Some possible domain engineering objectives could change, expand or diversify the business model of stakeholder organizations. In particular, domain models or information extracted from them can become products in their own right. For example, domain analysis results could be marketed as a comparative survey, a strategic marketing plan, a competitive analysis within the domain, a training instrument, a tool to aid in experts' collaboration and sharing of domain knowledge, a database or a reference guide. For companies unaccustomed to this perspective, the shift from

being product or service providers to information and knowledge providers may have profound repercussions.

5.1.2 Identify Candidate Objectives

In the previous task, *Determine Candidate Stakeholders*, we have identified the "cast of characters" for the project and modeled their overall roles, relations and interests. We deliberately "cast the net wide" to surface possible risks and identify opportunities for synergy in stakeholder interests.
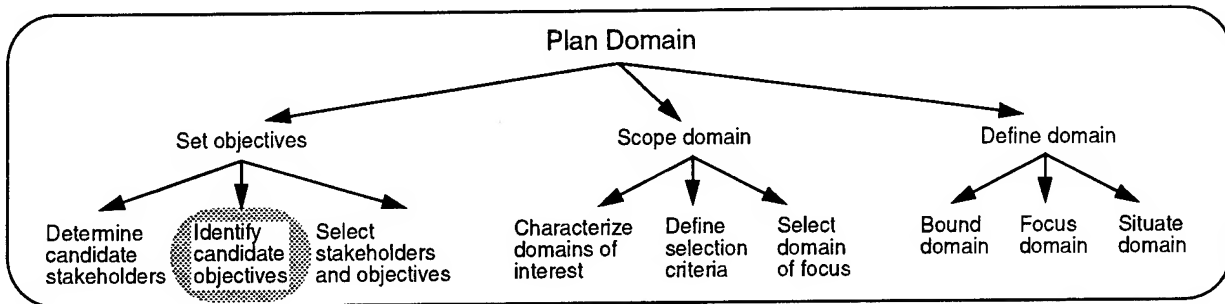
Exhibit 27. Identify Candidate Objectives Process Tree

We now want to generate a similar list of candidate objectives for the domain engineering project. A primary purpose of the *Identify Candidate Objectives* task is to ensure stakeholder interests are considered in identifying alternative objectives for consideration; in particular, to determine possible risks or barriers involved if objectives are chosen that conflict with certain stakeholder interests. For example, if practitioners in a selected domain perceive their domain knowledge as an important element in their job security, this could radically affect both objectives and data acquisition strategy. Another goal is to get people to consider a wider range of possible objectives for domain engineering. Closure on project stakeholders and objectives is obtained in the subsequent *Select Stakeholders and Objectives* task.

A key challenge in this task is to express desired objectives in forms more specific than general business objectives, but that do not get overly prescriptive in terms of assumed reuse technology approaches or domain selection. Just as stakeholder analysis is a natural way of introducing ideas like the "reuse marketplace" roles (asset creation, management and utilization) in concrete scenarios meaningful within the organization, so in this task the process of finding the right level of abstraction for stating candidate project objectives can lead to some broadening of perspectives.

Approach

The basic approach is to transform information from a variety of sources, including general STAKEHOLDER KNOWLEDGE and goals outlined in the PROJECT CHARTER, into specific candidate PROJECT OBJECTIVES. In order to guide this transformation, it is helpful to distinguish different levels and types of objectives.

Another key approach in this task is to map identified stakeholder interests to candidate objectives. This mapping might be very direct in first generating candidate objectives. But the later step of cross-checking all candidates with all interests (for at least key stakeholders) helps reveal some of the hidden agendas, potential points of conflict, and alternative strategies for realizing the same goals.

Levels of Objectives

In order to focus on the types of objectives that are to be identified in this task, it is helpful to distinguish different levels of strategic objectives for an organization. A simple typology, as illustrated in Exhibit 28 can be used to characterize a stated goal or objective with respect to two dimensions: scope of control (is this an objective we can commit to as a realizable result for the project?) and time span (is this an objective realizable within the time window of the proposed project?). Stakeholder interests identified in the last task may have ranged throughout these different levels. For this task, we want to identify concrete objectives for the project (bottom left quadrant).

Desired Outcomes		
Perceived risks and opportunities **Out of project scope of control**	**Mission** **Values** **Perceived Trends**
Within project scope of control **Measurable results**	**Follow-on actions from project** **Success criteria**

Scope of Control (vertical axis, from **Desired Outcomes** to **Actionable Decisions**)

Short term/tactical ◄────────► **Long term/strategic**

Time Span
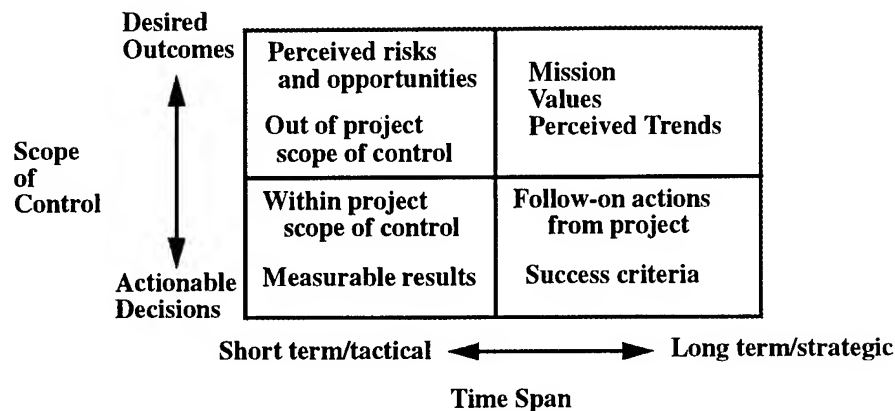
Exhibit 28. Characterizing Objectives vs. Interests

With respect to reuse and domain engineering objectives, these distinctions can be made more explicit. A number of broad objectives can be derived from the ORGANIZATION CONTEXT as documented in the STAKEHOLDER DOSSIER produced in the last task. These are "given" so to speak for the specific objectives to be elicited here. General organization goals/mission, based on sector, business model and core competitive areas, and the role of software in the organization, lead to what could be considered general software engineering interests (e.g., improving software process maturity, getting control of configuration management problems for product lines, etc.).

General reuse objectives are formed from the interaction of these software engineering objectives for the firm with other process improvement or learning goals (e.g., better codifying experts' knowledge in areas strategic to the firm). There may also be goals and interests tied to specific products and projects (e.g., pursuing a particular customer).

Domain Engineering Objectives

All the broader objectives described above typically lie behind the stated mandate (if one exists) within the PROJECT CHARTER. We now want to generate some specific domain engineering project scenarios that imply different objectives, and map them to the broader organizational objectives.

Example. Domain engineering PROJECT OBJECTIVES might include:

— Define standard specification models supporting domain-specific completeness and consistency checking at the requirements level.

— Reduce the total code size for a set of maintained systems, to reduce both resource requirements and maintenance burden.

— Codify and document veteran engineers' expertise in a domain.

Example. For the Persona scenario, some specific candidate objectives identified include the following:

— Provide a library of components that support application development in some vertical niche market (e.g., medical domain).

— Make all Persona products conform to PC look and feel standards; or

— Provide reusable capabilities not currently available through COTS systems APIs.

Each of these objectives can be traced back to stakeholder interests. The first echoes the broad issue of UI vs. vertical market business strategy within the firm. Jack Perricone has a particular interest in the medical area, to close a partnership with MMI on a lucrative contract in this business area. Note how easy it is for domain choices to "migrate" into the stated objectives. It will prove better to split this objective into aspects that will appear later as DOMAIN SELECTION CRITERIA.

Similarly, the second objective suggests two more specific domain engineering objectives: to "reuse" requirements from COTS packages in Persona products to maintain competitive viability; and to standardize the look and feel across Persona products, to produce a more unified product line.

Finally, the third objective gets at an essential element of a domain engineering objective. Persona cannot produce a spreadsheet interface and hope to compete with standard packages in this area. They have carved a business niche out of programming the "special" aspects required in customer UIs. This objective calls for identification of some UI-related area that has significant commonality across applications, but has *not* been successfully encapsulated as a COTS product. This may be possible because such products are generally engineered as end-user applications, not as components or configurable sub-systems to be incorporated into other applications. If Persona elected to take a "horizontal" rather than vertical approach, this might be a viable objective; on the other hand, risks of failing to find a domain satisfying these *strategic* objectives are high.

The broadest-level domain engineering objective can be documented in terms of the overall new organizational scenario envisioned where a domain model and/or asset base has been added to the picture that wasn't there before. We have seen how alternative scenarios of this kind begin to emerge out of the stakeholder roles and relations analyzed in the STAKEHOLDER DOSSIER. In this task we want to pull these out as explicit alternative scenarios.

Example. In the Persona scenario, two distinct scenarios emerged for where a potential asset base could be "housed" within the organizational structure. In one scenario, the asset base would be managed as an internal resource, with Maskara and Webonautics acting as asset utilizers in creating customer applications. In the other scenario, the asset base would be "downstream" of these groups; either a new group internal to Persona would take on the utilizer role and be the direct liaison with certain customer organizations; or, at least in theory, customer organizations (like MMI) might become utilizers of assets in their own right. This reflects one of the high-level decisions to be made in defining PROJECT OBJECTIVES.

Workproducts

■ PROJECT OBJECTIVES

A list of candidate objectives for the domain engineering project mapped to stakeholder interests and the PROJECT CHARTER. Each objective is qualified as to whether it is an incentive or disincentive for each stakeholder based on each of the stakeholder's interests. Exhibit C-3, PROJECT OBJECTIVES/STAKEHOLDERS INTERESTS MATRIX, contains a worksheet template that can be used to explore the impact of each objective on each stakeholder. Objectives need not be consistent, and, taken in aggregate, the set as a whole will not typically be achievable given PROJECT CONSTRAINTS. Note that PROJECT OBJECTIVES are not a control to this task.

When to Start

- Some stakeholders have been identified. This activity can be initiated as soon as some stakeholders have been identified in the *Determine Candidate Stakeholders* task. The STAKEHOLDER DOSSIER does not have to be complete.

- Charter is available. The PROJECT CHARTER must be available as a control used to determine objectives and how they relate to stakeholder interests. Although we are generating alternatives and not closing on decisions, the PROJECT CHARTER helps focus on objectives that are "in the ball park."

Inputs

- STAKEHOLDER DOSSIER. Stakeholder interests are one important source of PROJECT OBJECTIVES. Information about stakeholder organizations recorded in the dossier include the sector and business model, role of software in the enterprise, and current reuse practice and culture.

- STAKEHOLDER KNOWLEDGE. Supplementary information gained from stakeholders from which PROJECT OBJECTIVES can be elicited.

Controls

- PROJECT CHARTER. Used to assist in determining how objectives relate to stakeholder interests. The PROJECT CHARTER is also one source of objectives, by translating the charter into objectives that encompass its intent.

ORGANIZATION CONTEXT is also a critical factor in determining objectives. However, this information should be reflected in the STAKEHOLDER DOSSIER.

As with the previous task, PROJECT CONSTRAINTS are *not* a control on this task because the task involves the development of a list of *unfiltered* candidate objectives. In this task, it is important to record all objective that are incentives to key stakeholders, even if they are contrary to PROJECT CONSTRAINTS. Constraints will be used to filter the set of candidates objectives in the *Select Stakeholders and Objectives* task. Objectives that are not chosen but are incentives to key stakeholders are still useful to document, because they help flag project risks.

Activities

➤ <u>Set context for objectives</u>

Determine the categories of objectives that will be considered when identifying PROJECT OBJEC-
TIVES. Developing these categories can ensure that important categories of objectives are not
missed when deriving candidate objectives, and that reuse-specific objectives are the focus.

In order to map candidate PROJECT OBJECTIVES to stakeholder interests, a hierarchical model of
objective categories can be created spanning:

- General business objectives for the organization,

- Software engineering objectives, and

- Overall software reuse adoption objectives.

For each stakeholder organization, this hierarchy can be used to classify/search for objectives in
each category for the organization. Stakeholder interests can be linked to objectives at various
levels within the hierarchy. (A simpler alternative is the "quadrant" model illustrated in the
Approach subsection above as Exhibit 28.)

➤ <u>Construct reuse-based engineering scenarios</u>

Build concrete scenarios about how successful reuse could be performed in the project context,
based on the PROJECT OBJECTIVES and stakeholders. Key benefits of this activity are to:

- ground planning in specific, tangible outcomes,

- allow creative thinking in support of project planning, and

- motivate the team by building a shared vision of project goals and asset users.

Brainstorm potential end uses for domain engineering products including domain planning prod-
ucts, domain models, architectures, and assets. Challenge tacit assumptions with counter-scenar-
ios; if people are assuming that the only useful outcome of domain engineering is reusable code
consider scenarios where other work products like requirements are the focus of the effort.

➤ <u>Derive candidate objectives</u>

Drawing from the PROJECT CHARTER, characterizations of stakeholder interests in the STAKE-
HOLDER DOSSIER, and other STAKEHOLDER KNOWLEDGE, identify specific candidate PROJECT
OBJECTIVES for the project.

Candidate objectives can be identified by the following means:

- The PROJECT CHARTER can be translated into a set of objectives that encompass its intent.

- Objectives can be identified that relate to specific stakeholder interests in the STAKEHOLDER
  DOSSIER. The hierarchy developed in the previous activity can be used to help ensure that
  objectives are identified in all categories.

  What project outcomes would constitute success for this stakeholder, with respect to this
  interest? What outcomes would be perceived as undesirable? For example, for a product divi-

sion concerned with time to market, one candidate objective might be to create an asset base that would significantly reduce development time for new products in that division.

- Project members can suggest PROJECT OBJECTIVES.

Project objectives can be couched in terms of organizational goals and/or systems. The following list contains some sample project objectives:

— Improve programmer productivity, as measured by cost savings in new development of system Y.

— Minimize redundant software engineering efforts.

— Improve software quality.

— Codify departing expert knowledge.

— Consolidate product line P.

— Eliminate costly requirements anomalies in systems by comparing project requirements to a generic set of requirements that fits the domain.

— Obtain uniformity in user interface in product line Q.

— Identify new markets for components of existing systems.

— Increase system maintainability.

— Reduce system cost.

— Reduce time to market in creating new applications.

— Reduce risk.

— Assist in the reengineering of system X.

➤ Map objectives to stakeholder interests

Map each candidate objective to each stakeholder interest identified in the STAKEHOLDER DOS-SIER, based on information about stakeholders contained in the STAKEHOLDER DOSSIER and other STAKEHOLDER KNOWLEDGE. Although stakeholder interests were the source of many of the candidate objectives, it is now necessary to explore the impact of each objective on other stakeholders. Exhibit C-3. Exhibit 25 shows an example of a filled out PROJECT OBJECTIVES/STAKEHOLDERS INTERESTS MATRIX, contains a worksheet template that can be used for this activity.

Allow for conflicting motivations for stakeholders. Certain outcomes may present both an incentive and a disincentive for a given stakeholder. For example, an engineering project may desire to move to a standard set of components, yet be concerned that the effort to create these components

will have to be funded by the project. This exploration may lead to the discovery of new relationships between objectives and stakeholders.

Candidate Objectives	Stakeholder Interests											
	Connell			Morro	Webonautics		MasKara		Persona		Perri	MMI
	Collaboration	Gov't Security Standards	Combat graphics perceptions	Attract best technical people	Wide audience	Distrib system	Consistency with gov't work	Reduce production costs	High-profile hit	Make a splash to attract talent	Partner with MMI	Get contract with CDC
Produce platform independent library for WEB and CORBA	!	0	+	+	!	!	+	!	-	+	X	X
Library support for vertical market medical law	0 0	+ 0	- -	0 0	- -	- -	- -	- -	+ +	+ +	! X	! X
Make all persona products conform to PC look and feel standards	0	0	-	-	+	-	X	X	-	-	+	+
Provide capability not currently available from COTS systems through API	0	0	!	+	+	0	0	0	0	+	0	0

Exhibit 29. *Example Excerpt*: Project Objectives/Stakeholders Interests Matrix

➤ Map objectives to charter

As a final validation, map candidate objectives to the PROJECT CHARTER to determine the fit of various objectives to the overall charter. Candidate objectives are mapped to the charter in a similar manner as they were mapped to stakeholder interests in the preceding activity.

When to Stop

- Candidate objectives are defined. A set of candidate PROJECT OBJECTIVES has been explicitly defined.

- Objectives are mapped to charter. Each objective is mapped to the PROJECT CHARTER.

- Objectives are mapped to stakeholder interests. Each objective is mapped to the stated interests of stakeholders.

As a validation, we can check that each *key* stakeholder has at least one objective identified that would be a plausible basis for commitment. If we have a key stakeholder and no candidate objectives that "make the case" there is no point in going on to the selection task. Iterate back as needed to find new interests, new objectives to map to identified interests, or an alternative scenario where the key stakeholder is no longer key. Otherwise conditions are not in place to continue the planning effort.

Guidelines

- **Do not base objectives solely on project requirements.** Identify at least some objectives that do not merely reflect external mandated requirements or constraints on the project.

- **Do not unduly constrain objectives.** Do not constrain objectives by making a premature commitment to an approach or technology (e.g., commitment to an object oriented design approach). Specific approach and technology approaches should be made during the Asset Base Engineering phase, once the domain has been modeled and analyzed. *Do not* allow this activity to gravitate too strongly towards specific implementation strategies. A good method of mitigating this risk is to brainstorm alternative implementation strategies for any given PROJECT OBJECTIVE and to allow these alternatives to remain as open possibilities.

- **Do not confuse objectives with domain choices.** A typical mistake is to confuse overall domain engineering objectives with the choice of domain as the project's domain of focus. Objectives should not directly reference a specific domain, unless these constraints are imposed by the organization. Domain selection will be performed later, in the *Scope Domain* sub-phase.

- **Address non-technical objectives.** Do not exclude or avoid non-technical candidate objectives. A primary purpose of this task is to ensure that PROJECT OBJECTIVES are realistic. This means getting the real issues out in the open. If possible, document and consider even politically sensitive objectives that may seem to offend the "purity" of technical criteria. (If necessary, leave their source unrecorded to make the issues less confrontational.)

  Example. The president of Persona, Chris Connell, is worried about the separate technical agendas of the two groups, Maskara and Webonautics. This was part of his motivation for approving the pilot project. But these concerns are based in part on the personalities of the heads of the two divisions, and their occasional difficulties at "playing from the same sheet of music." Neither would take well to a management mandate to "cooperate" and promoting the project in this light would probably be the kiss of death!

  Yet this stakeholder interest leads to distinct objectives that should be considered. For example, to create a cooperative team interaction between the two groups it might make sense to pull members from both teams onto the project, then rotate them back into their respective divisions to serve as advocates for the commonly developed assets. This personnel strategy may not be essential to realizing the other technical objectives of the project, but may be critical to addressing this management concern. Some balance must be found between leaving this objective unstated and inciting resistance by publishing it inappropriately.

- **Consider both perceived incentives and disincentives** for each stakeholder, and to consider implications of both positive and negative outcomes in each case. For example, an engineering group may perceive a mandated standard architecture as a risk because it will impose severe constraints on performance. A project that assessed the domain and judged such an architecture infeasible might be considered a positive outcome by these stakeholders. So might an outcome where a more flexible approach to a domain architecture was recommended.

- Identify conflicting objectives. Some potentially conflicting objectives *should be* identified. Be wary if only consistent objectives emerge from the *Identify Candidate Objectives* task. The process should reveal some alternatives and trade-offs; otherwise some relationships between objectives and stakeholders may not have been considered. this may also be a sign that the group has tried to come to closure too soon.

- Identify technology and business objectives. Conflicting technology and business objectives are a classic source of problems in domain engineering projects. There are typically both software developers and technology enthusiasts who advocate the use of certain technology within an organization. There are also stakeholders concerned primarily with addressing business objectives. If both technology and business objectives are selected for the project, clear priorities must be set.

- Consider domain modelers' objectives. Domain engineering project members are project stakeholders. Consider objectives, including personal objectives, of those who will partici-pate in the modeling effort. For example, "performing domain analysis to learn about a new and interesting domain" is a legitimate candidate objective for a project member (although it may have lower priority for the overall project than company business objectives). In general, *learning objectives* must be considered along with "productivity" objectives that yield tangible results.

- Watch out for technical idealism. A domain engineering project may be seen as the chance to implement the elusive "elegant general solution" that time constraints and pragmatic con-cerns usually prevent from being realized. This can present both opportunities and risks. It may be a strong motivator for engineers to participate in the project. But they can also get very attached to using the project as a way of advancing their own technical agendas. They need to be willing to participate in the modeling *process* itself, which implies an openness to multiple perspectives.

5.1.3 Select Stakeholders and Objectives

We have completed two "descriptive" tasks where we identified candidate stakeholders and objec-tives, linked together via a strategic picture of how stakeholder interests coincide, conflict and map to candidate objectives. The primary purpose of the *Select Stakeholders and Objectives* task is to select specific project stakeholders and objectives that are consistent with the PROJECT CHAR-TER and attainable given PROJECT CONSTRAINTS with an acceptable level of risk.
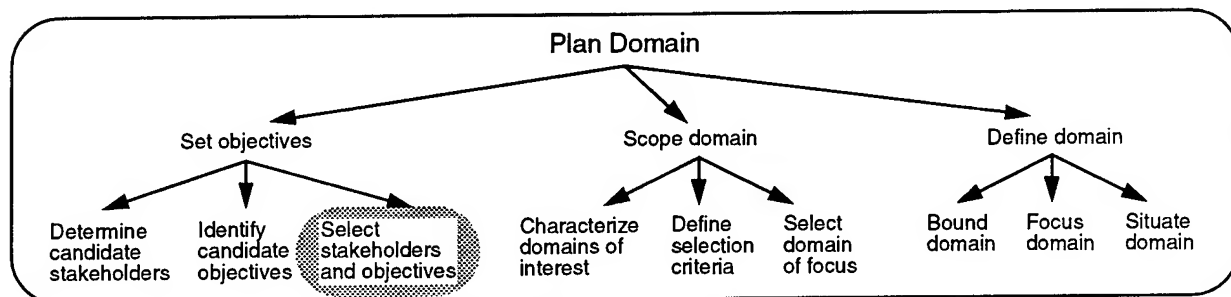
Exhibit 30. Select Stakeholders and Objectives Process Tree

The objectives selected should satisfy key stakeholder interests and maximize chances for adop-tion of domain engineering results and technologies within stakeholder organizations. Once objectives and stakeholders are selected, clear project expectations should be established with all stakeholders, including project members.

One challenge of this task is that it involves both a complex cost/benefit tradeoff analysis and, in effect, a team-building task. Generally, we need to consider the strategic payoff to multiple stakeholders, related via tiered "value networks" of technology producers, brokers and consumers. Specific domain engineering scenarios often introduce new "tiers" into this value network. Projecting relative costs and benefits for this new configuration, much less comparing it with the status quo or with alternative configurations, does not make for a simple cost-benefit analysis. From a formal standpoint, both the final set of stakeholders and the final set of objectives can be dynamically "tuned" to arrive at an optimal scenario. [2]

At the same time, a process that produces a viable scenario for moving forward but alienates key players in the process will not be successful. Key players don't include only decision-makers such as high-level managers. There may be a respected technical staff person whose buy-in to the project is critical to establishing its credibility within the engineering groups that must eventually use the assets. Obtaining commitment from the right group of people is a major milestone in this task, and for the *Set Objectives* sub-phase as a whole.

Approach

The ODM approach to this task include the following key ideas.

Stakeholders and objectives are selected together. This is the rationale for the process model of the sub-phase as a whole. Choices on one side dynamically affect choices on the other side. The goal is to find a stable and robust configuration, one where:

— success does not depend entirely on achieving one (possibly high-risk) objective, and

— project viability does not depend entirely on the buy-in of one (possibly fickle) key stakeholder.

The "discipline" of deferring closure to this task has actually served to create social interactions between stakeholders that increase the chances for successful negotiation of shared objectives.

Because stakeholders and objectives can be selected strategically in one process, it is possible to find an optimal balance between ambitious and too-modest objectives. This is perhaps the key tradeoff in this task. The project needs to be big enough to be "on the map" and not treated as a "toy effort." It needs to be aligned with real stakeholders who intend to really make use of the results as opposed to an exercise to trial-run the methods and tools, a common approach to a first pilot. (Without this tie-in to real business goals, a trial run is not really a trial, because the stakeholder picture is artificial.)

On the other hand, the project needs to be small enough to "fly in under the radar." Domain engineering initiatives are attempts at organizational change. Organizations (and people) have effective protection mechanisms to resist change. One classic strategy is the "set-up" that tempts a pilot project into vulnerability by getting the project team to over-commit to results, based on commitments of resources that will invariably be reduced (unlike the corresponding expectations). The Catch-22 is that a project can wind up "too big" and "too small" simultaneously!

[2.] This is, interestingly enough, analogous to the general methodological problem of domain engineering as lacking a single-system point of reference. Thus, in performing this task, as with many other tasks in the *Plan Domain* phase, planners are simultaneously dealing with the necessary organizational issues and practicing formal modeling skills that will be put to use in the technical domain, in the *Model Domain* phase.

The notion of "commitment" in ODM therefore has a particular emphasis. It involves simultaneous trade-off analysis in terms of selecting stakeholders and objectives, and direct intervention in the relations and expectations of the organization, as necessary, to open a "window of opportunity" for a project success.

A general rule of thumb is that such projects tend to be smaller in scope than people first anticipate. Since a domain engineering project integrates results and/or requirements from multiple projects, if the scope of the project is comparable to a typical project (in the organization's terms) it is probably in the "too big" category, and will present an unacceptable level of risk.

It is best, therefore, to define very incremental results, with deliverables defined at each step that have "collateral pay-off" for various stakeholders, and maintain the ability to rapidly re-scope the project as necessary as obstacles and opportunities arise. It is also useful to know up front the many "hidden" costs and resource requirements (e.g., access to domain experts) that need to be contracted for up front to ensure project success.

Workproducts

■ PROJECT OBJECTIVES

A list of the selected PROJECT OBJECTIVES linked to the PROJECT CHARTER and project stakeholder interests. Success criteria for each objective are also included, and optionally measures that can be used to determine whether each success criterion was met.

It can be helpful to structure the objectives to clearly show:

- Current state of the targeted stakeholder picture;

- Desired end-state at completion of the project (near term objectives described as an integrated scenario);

- Specific criteria or measures to apply in the end-state configuration to evaluate project success (e.g., that certain specific groups will be performing the asset utilizer role with some level of usage);

- Optionally, an additional end-state scenario that describes longer-term desired changes, beyond those to be introduced by the project specifically.

Exhibit C-4, PROJECT STAKEHOLDERS/OBJECTIVES SUMMARY MATRIX, shows a worksheet template that can be used to map objectives against each stakeholder interest as a cross-check to show the level of commitment to each objective.

■ PROJECT STAKEHOLDER MODEL

The PROJECT STAKEHOLDER MODEL contains a list of the stakeholders whose interests will be satisfied by the selected PROJECT OBJECTIVES. The PROJECT STAKEHOLDER MODEL includes stakeholders from the STAKEHOLDER DOSSIER who are still of interest to the project and any new stakeholders that emerge as a result of selected objectives. Any relevant information from the dossier, such as descriptions of roles, relationships, interests, enablers and barriers, and assessments of attitudes towards reuse and reuse practices, can be referenced as needed in the model. If the roles of any stakeholders will change if the domain engineering project is successful, these role changes should also be described in the PROJECT STAKEHOLDER MODEL. For each stakeholder, the model should specify the following:

- Intended role on the project, including relations with other stakeholders;

- The anticipated benefit of the project to the stakeholder (their payoff);

- Their responsibilities (which can include support within management from sponsors, committed resources, and direct participation);

Clearly, some linkage information between stakeholders, interests, and objectives must be derivable from these two workproducts. Where the information is stored depends on details of the representations selected, which are not specified as part of the core ODM process.

When to Start

- Candidate stakeholders and objectives have been identified. Since the *Select Stakeholders and Objectives* task involves trade-off analysis between stakeholder interests and objectives, both the *Determine Candidate Stakeholders* task and the *Identify Candidate Objectives* task must be completed before the *Select Stakeholders and Objectives* task begins.

Inputs

- PROJECT OBJECTIVES. Candidate objectives are selected as objectives for the project. Although a workproduct of the same name is an output of this task, the inputs are candidates only while the output objectives have been committed to by the team and the stakeholders.

- STAKEHOLDER DOSSIER. Candidate stakeholders are selected as project stakeholders.

Note that in addition, this task will involve specific *interactions* with stakeholders around decisions. This is not gathering of additional information, but rather negotiation around decisions.

Controls

- PROJECT CONSTRAINTS. Restrictions imposed by organization and market conditions beyond the project scope. Constraints may include implicit expectations about project results, perceived risks, and other factors that could compromise the success of technically sound results.

  Constraints may include the following:

  — Technology constraints

  — Constraints on domain selection

  — Available resources, as well as constraints on resources that can be tapped. For example, there may be a pragmatic constraint that the domain engineering effort cannot have any impact on the schedule or resources available for an application engineering project of high priority. Such a constraint can significantly shape viable objectives for the project.

  — Restrictions on the scope of organization boundaries that the project can cross. For example, a domain engineering project in a software maintenance organization may have restricted access to organizations that developed application systems in the domain.

  PROJECT CONSTRAINTS will be iteratively interpreted relative to PROJECT OBJECTIVES as issues are raised.

The PROJECT CHARTER is *not* a control to this task. Relevant information should have been incorporated into the candidate PROJECT OBJECTIVES in the *Identify Candidate Objectives* task.

Activities

➤ Assess each objective

Given available resources and other PROJECT CONSTRAINTS, assess each candidate project objective in terms of:

- level of risk that the project could not obtain the objective,

- estimated level of effort, and

- anticipated pay-off, in terms of the set of stakeholder interests most directly addressed.

➤ Perform stakeholder trade-off analysis

Key stakeholders (whose buy-in and commitment are deemed essential to the overall success of the project) were already chosen in the *Determine Candidate Stakeholders* task and identified in the STAKEHOLDER DOSSIER. Other stakeholders in the STAKEHOLDER DOSSIER (who are not key stakeholders) are now prioritized, by performing a trade-off analysis on these stakeholders.

➤ Assess stakeholders and objectives in combination

This is the key activity in the task. The central idea is that you are selecting a set of stakeholders as project stakeholders and a set of objectives as project objectives simultaneously and iteratively. (A comparison could be made to forming a club: Who will be the members? What are the topics?)

A number of supporting methods and techniques can be used to manage the interactions in this process. Besides techniques for Stakeholder Analysis described in Section 8.1 of this guidebook, techniques for conflict resolution and mediation and negotiation skills can be useful here.

The risks in this activity include deadlock over decisions (conflicting non-negotiables), non-optimal opportunity analysis so that some potential benefits to stakeholders are not seen and capitalized upon; and the problem of "false buy-in": seeming acceptance of objectives and commitment which can later give way to resistance and even deliberate sabotage. The following paragraphs provide some suggestions for strategies to apply that can minimize these risks, but these will need to be worked through as appropriate in each situation.

- *Look for "common ground" objectives.* Identify the clear wins, objectives for which it appears easy to get consensus from key stakeholders. Having identified these, however, don't consider them closed yet. You may need some flexibility to work pass other impasses.

- *Look for impasses.* Next, get the bad news. Look for points that appear to be potential impasses, such as conflicting interests between two key stakeholders, anything that might prevent movement. For each stakeholder, objectives can be roughly sorted into "must haves," "negotiables" and "can't haves." A typical impasse is where one key stakeholder's "must have" is another's "can't have."

- *Work each key issue.* The long-deferred point of issue resolution that may involve getting someone to shift a position.

    — If both parties can shift from a key (non-negotiable either pro or con) to a negotiable sta-

98

tus for the objective and still stay "in the game" (not drop out as stakeholders), this is the best outcome; it leaves the later negotiation most free.

— Can one side shift to a negotiable? This is almost as good an outcome, since both stakeholders remain in the game.

— If neither side can shift, consider the scenario of shifting one or the other (*or both*) stakeholders from a "key" to a "negotiable" position. (Of course, if this is the project sponsor or division manager it is not so easily done; you may get shifted from indispensable to dispensable as a consequence!) Can you do the project without them? If so, *is it the same project?*

  If both stakeholders are critical for the success of the project as currently identified and the positions can't be shifted, declare the project a non-starter and terminate it. You must be willing to take the results of the stakeholder analysis seriously. If you believe the analysis and have reached this point, then it would not make sense to continue. (It is also true that proposing to terminate the project is the last way to find out if positions are really unshiftable; but you must be prepared to really end it if they are not. People will know if you're bluffing.)

• If you've read this far, you got past the show stoppers. (Congratulations!) The next step is to see whether the agreed upon objectives, and the stakeholders still in the game, provide an adequate basis for a workable project. This is where considerations of the scope of the project come into play. You may need to be able to build synergistic motivation for enlarging the scope via additional objectives and technical goals in order to have a project of the right size and visibility to have credible impact.

  However, keep in mind that to tune the PROJECT OBJECTIVES to maximum conditions for success, you may also need to consider *downscoping* the proposed project at this point. This could include not accepting all offered resources, in order to propose a smaller-scale pilot project where risks can be controlled and expectations managed.

  The PROJECT OBJECTIVES/STAKEHOLDERS INTERESTS MATRIX (depicted previously in Exhibit 29) is a useful aid in looking for these potential points of synergy.

• The project must be feasible given the available PROJECT CONSTRAINTS. This is the point where these constraints should be considered most carefully. Once again, if constraints are insufficient look for potential ways to get synergistic support.

• As a final step, once the project is feasible, optimize the plan given all the materials described above. This allows for a final tuning of the project scope, deliverables and schedule.

➤ Select and document objectives

PROJECT OBJECTIVES are selected based on the interests of key stakeholders and other stakeholders with high priority. The objectives selected and the project stakeholders whose interests are satisfied by the objectives should be documented in the PROJECT OBJECTIVES and PROJECT STAKEHOLDER MODEL.

The objectives selected are documented in the PROJECT OBJECTIVES.Objectives should be documented clearly and concisely, in language that will be easily understood by new project members over the lifetime of the project. The objectives also serve as a statement of expectations about the project for upper management.

Project stakeholders whose interests are satisfied by the chosen project objectives are documented in the PROJECT STAKEHOLDER MODEL.

➤ Validate objectives

Objectives are also validated in terms of the following criteria:

- Minimal overlap between objectives to ensure the list is concise and understandable.

- Interests of key stakeholders are satisfied by some subset of objectives.

- No project objective is in serious conflict with interests of key stakeholders. Any direct conflicts between interests of key stakeholders and project objectives must be resolved or at least flagged as a risk. Resolving key conflicts may require involvement and decisions beyond the immediate control of the project. Less critical conflicts and trade-offs can be resolved at a later stage of the project by tuning PROJECT OBJECTIVES. The goal is to get a stable set of objectives that the subsequent Planning tasks can be based on.

- The set of selected objectives constitute a sufficient guarantee of success for the project (i.e., if all objectives are satisfied the project be considered a success).

Objectives should also be mapped against each stakeholder interest as a cross-check to show the level of commitment to each objective. Exhibit C-4, PROJECT STAKEHOLDERS/OBJECTIVES SUMMARY MATRIX, shows a worksheet template that can be used for this activity. In this worksheet, stakeholders who have been selected as project stakeholders are now repartitioned into key and other stakeholders based on the objectives chosen for the project. For each stakeholder, based on all of the stakeholder's interests, cells are filled in to rate each stakeholder's overall level of commitment to each objective. Information that was collected in the STAKEHOLDER DOSSIER can be used to help determine anticipated commitment level.

> *Example.* For the Persona scenario, Exhibit 31 shows an excerpt of the final selected objectives and stakeholders. One point of concern is Jack Perricone's buy-in to the project. Clearly, the highest priority objective (platform-independent library for Web and CORBA) is a disincentive for him, and his main "hot button" (partnership with MMI) is the lowest-priority objective. The underlying conflict is the directing of the project towards a horizontal rather than vertical emphasis.

> After some negotiation, the following shifts take place (not reflected in the example excerpts). The objective "Make Persona products conform to PC look and feel standards" is added to the objectives, another objective for which Perricone has strong interest. A strategy discussed is to consider using the outside consultant, ODM Inc. to do some comparative survey of COTS interface conventions for the domain selected. This also increases ODM Inc.'s "stake" in the project, since there will be more public-domain results that can be shared as technology demonstrations. (This added incentive is reflected in a discounted consulting price negotiated by Francesca Morrow). After discussion, Perricone also shifts his own position so that he sees the strategic value to the company of the objective "Provide capability not currently available from COTS systems..." The overall "balance" of the stakeholders and objectives is now reasonably sound.

Project Stakeholders (Selected & Prioritized)		Project Objectives (Selected & Prioritized)			
		Produce platform independent library for both Web and CORBA	Assist in production of systems with functionality familiar from popular systems	Provide capability not currently available from COTS systems through API	Close partnership contract with MMI and CDC
Key Stakeholders	C. Connell	+	0	0	+
	F. Morrow	+	+	+	+
	Webonautics	+	0	+	0
	Maskara	+	0	0	0
	Persona	0	0	+	+
	J. Perricone	-	0	0	+
Other Stakeholders	MMI	-	0	0	0
	AFDSO	0	0	0	-
	ODM	0	0	0	0

Exhibit 31. *Example Excerpt:* Project Stakeholders/Objectives Summary Matrix

➤ Establish success criteria, measures and risk mitigation strategies

Identify success criteria that can be used to determine whether each objective was met. Success criteria can be decomposed into specific measures that can be used to evaluate whether each success criterion was met. These measures can be fed into the overall project plan, to help determine what kind of measurement data needs to be collected to demonstrate that particular objectives have been met, and when review and evaluation activities should be scheduled to ensure that progress is monitored at reasonable intervals.

Suppose that one project objective is to estimate the level of effort that will be required for subsequent domain engineering projects in the same organization. This implies that measures of the level of effort will be a critical output of the current domain engineering project. If the current project involves creating infrastructure that will be utilized unchanged on subsequent projects, sufficient measurement data must be captured to allow infrastructure development effort to be deducted from the total effort to determine the level of effort for domain engineering by itself.

➤ Obtain stakeholder commitment

Since objectives are the project's contract with stakeholders in the organization, reviews of the objectives are held with these stakeholders. At these reviews present the objectives, indicate the motivating interests, and describe risk mitigation steps that address possible concerns. Stake-

holder sign-off must be obtained on the acceptance of each included objective, and possibly on the basis of the *absence* of other objectives.

Buy-in from application groups that could potentially use assets produced by the domain engineering project is of primary importance. Present reuse-based engineering scenarios to show these stakeholders how domain engineering products could be used. However, don't attempt to get commitments for certain "quotas" of reuse levels. Assets must be reused based on their own technical merits. While reuse policies and incentives showing support of management can encourage reuse, reuse mandates may be counter productive.

Obtaining stakeholder commitment is an iterative process. If a set of key stakeholders cannot be resolved with a set of project objectives, iterate through the task again. Use feedback from stakeholder reviews to update the PROJECT STAKEHOLDER MODEL and PROJECT OBJECTIVES. Hold further stakeholder reviews until commitment is obtained, or a determination is made to terminate the project. If a few iterations cannot resolve the impasse, it may be best to stop the project.

When to Stop

- Objectives are defined. A set of domain engineering PROJECT OBJECTIVES has been defined and agreed upon by the project members.

- Project stakeholders have been identified. For each stakeholder, the anticipated benefits and commitments required are documented in the PROJECT STAKEHOLDER MODEL.

- Commitment has been obtained. Key stakeholders have seen and approved the PROJECT OBJECTIVES. Their key interests are satisfied by the selected objectives. Interests and objectives of each key stakeholder with respect to the project have been resolved for consistency. Necessary resources have been committed to the project.

If above conditions have not been achieved after several iterations:

- Stop the project. If a consistent set of key stakeholders can not be resolved against proposed project objectives, it may be best to not move ahead with the project.

Guidelines

- Objectives can be specific to systems. E.g., an objective is to build assets that will assist in the reengineering of a specific system X, or that will contribute to cost savings in the new development of system Y. These are real technical (as opposed to learning or process improvement) objectives for the organization, but are *not yet* domain-specific. They still leave free exploration of what domains intersect these systems of interest, the focus of the *Characterize Domains of Interest* task.

    Example. In the Persona scenario, the strategic picture for the project relies on Jack Perricone's buy-in, and this depends on the connection with MMI being specifically addressed. This focuses the project on some domain that will intersect the system being build with MMI.

    On the STARS Army Demonstration project, the project objectives were linked closely to a charter for reengineering a specific system within their set of maintained systems.

    Similarly, project objectives could be couched in terms of organizational goals. For example, an objective might be to "build assets that will be used by groups A and B." (This could even extend towards "process" domains rather than straight software domains.)

Example. In the Persona scenario, implicit in the objective to build assets that span Web and CORBA platforms is an organizational objective to get assets shared by both the Webonautics and Maskara groups. This objective traces directly back to Connell's concern about shared vision within the company.

- <u>Defer technology choices when possible</u>. Known constraints on technology decisions should have been reflected in the PROJECT CHARTER. Try to state domain engineering project objectives in a form that doesn't unduly prescribe technology choices that are not mandated. For example, an objective to build a library of Ada code components to support the selected domain has a specific implementation approach (Ada) embedded within it. This implementation approach may or may not prove appropriate for the domain eventually selected. Informed technology choices should be made during the *Engineer Asset Base* phase, based on the characteristics of the domain.

In some cases, valid constraints on technology selection will appear as scheduling constraints rather than a preference for a certain technology. Interdependencies between the domain engineering project and other engineering projects may create pressure to make some technology choices rapidly (e.g., the choice of architecture representation). Conversely, dependencies on technology choices that will be made by external groups may introduce some uncertainty into project planning.

5.2 Scope Domain

In the previous sub-phase, *Set Objectives*, a set of explicit PROJECT OBJECTIVES was selected and aligned with the interests of a specified set of project stakeholders. As far as possible, the defined objectives have been defined independent of a selected domain. The primary purpose of the *Scope Domain* sub-phase is to make a DOMAIN SELECTION for the project that is in alignment with these PROJECT OBJECTIVES. The selected domain is called the *domain of focus*.
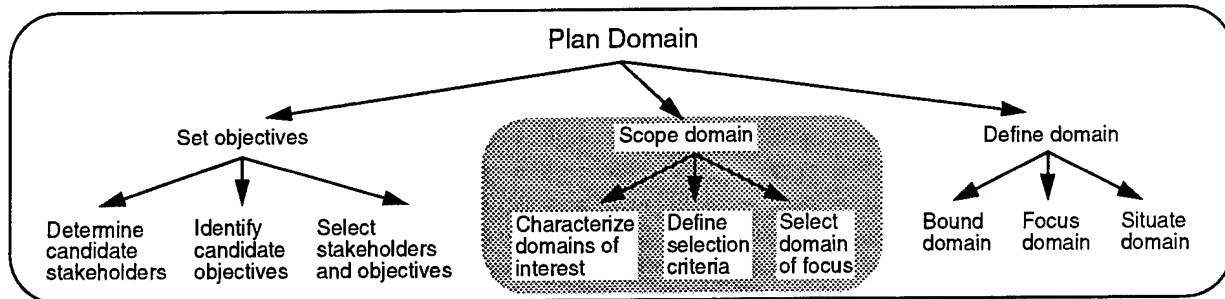
Exhibit 32. Scope Domain Process Tree

The major objective of this sub-phase is to probe for potential domains thoroughly enough that possible risks are identified, and an optimum selection is made. *Selection of the domain of focus is the single most important strategic decision of the domain engineering project.* It is the decision that has the most impact on the overall success of the project. Yet it is not a simple decision to make, in part because the domains that may be the best areas of focus for a given organization may not correspond to the familiar ways people in that organization divide their work up into systems and organizational settings.

Thus the real added value of this sub-phase, and its primary challenge as well, is to help promote a "domain's-eye" view of the organizations and systems within the ORGANIZATION CONTEXT. People are accustomed to viewing the business environment in terms of existing organizational structures and systems (established project groups, product lines, etc.) But *knowledge* lives within the organizational setting within patterns of communication that cross all of these established boundaries. The most fruitful domains to address are those that codify knowledge areas in which the company has both competency and strategic interest. This can involve grouping systems and functional areas in ways unfamiliar to stakeholders. These less obvious domains sometimes turn out to have the highest potential for payoff.

With this domain-oriented re-envisioning of the systems and work groups within the ORGANIZATION CONTEXT, more is achieved than the selection of a single domain for the project. Multiple domains of interest are identified that may have interactions with the domain of focus that will need to be investigated later in the process or may become the focus of subsequent efforts.

Approach

The term *domain of focus* provides insight into the nature of the *Scope Domain* sub-phase. Domains are abstractions that group particular sets of systems, or areas of functionality within systems, in ways that allow strategic reuse of assets within the domain scope. Domain relationships are not analogous to system, subsystem, and module relationships. Domains can overlap and even enclose other domains. Scoping a domain is therefore not like choosing an item out of a catalog; it involves deciding where to most strategically draw domain boundaries. In the following paragraphs, we will discuss some critical tradeoffs in the ways that domain boundaries can be drawn within the *Scope Domain* sub-phase.

Exhibit 33. Scope Domain IDEF$_0$ Diagram
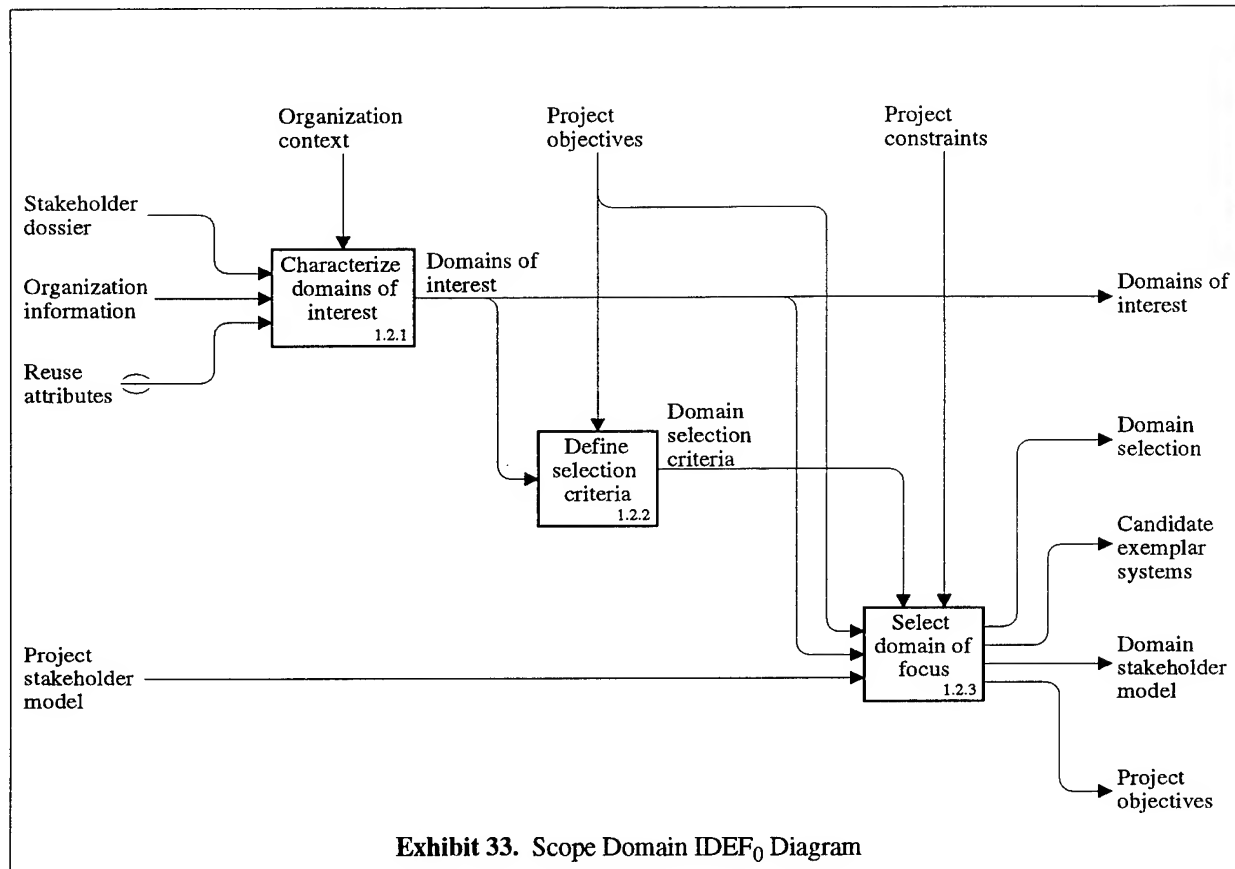
In ODM domains are always defined relative to specific organizational stakeholders, and relative to a specific set of systems of interest within the ORGANIZATION CONTEXT. Systems of interest are used to characterize DOMAINS OF INTEREST in terms of their intersection with these systems. Systems of interest include any applications (software, systems, organizational processes) "of interest" to stakeholders identified in the STAKEHOLDER DOSSIER. "Of interest" can mean many things: the system is under development, being maintained, being assessed for possible acquisition, or being tracked as an industry competitor. In a development environment, systems of interest include both legacy systems and anticipated new systems. In a maintenance environment, systems of interest include the inventory of systems under control of the organization initiating the domain engineering project. We will see how ODM formalizes many ambiguous notions of domain by relating domains to systems of interest.

Native and Innovative Domains

Even though most software organizations may not use the word "domain" they all have established ways of looking at aggregate functionality across individual systems. These groupings are typically at a high level, e.g., "business areas" or "lines of business" for a large and diversified company, "product lines" and "technology platforms" for a product-oriented company, etc.

In ODM we want to capture these "native" categories for domains, be they at the system or subsystem level. First, these certainly may turn out to be promising areas for a pilot domain engineering project. Second, we will need to understand this frame of reference to suggest anything different. However, a key element of the ODM approach is to look beyond these familiar domains to identify possible domains that may cross existing boundaries. These innovative domains can have several advantages.

- First, they can reveal commonality that has been "hidden" because people have not looked at systems from a unifying perspective. If this commonality can be captured in reusable assets, significant benefits may be achieved.

- Second, innovative domains can help to decompose large-scale domains or business areas into areas that are more tractable in scope for pilot projects. This is basically but not entirely a risk reduction strategy. The larger the domain, the more likely it is that there will be many coincident contextual dependencies embedded in the architecture and various design decisions of the components in the domain. More focused domains take more work to tease out the contextual assumptions and potential areas of application; but the payoff is a more flexible set of reusable assets.

- Finally, the systematic exploration of innovative domains is itself a way of getting people to view work practice differently; it "shakes things up" in useful ways. This is where domain engineering can be viewed as much as organizational intervention as technical work, and this is a recurring theme throughout the ODM life cycle.

Vertical and Horizontal Domains

A domain is a specified scope of applicability for a set of reusable assets. In practice, a domain will usually be an area of functionality that occurs across a set of systems. The terms "horizontal" and "vertical" occur often in domain analysis literature. The terms are used by analogy to vertical and horizontal integration in business processes: vertical domains are domains of "whole systems" for solving certain classes of real-world problems. Examples of vertical domains might include banking applications, flight simulators, command and control, factory automation systems, or telephony systems. "Horizontal" domains are usually smaller-scale utility domains that occur across a number of different types of applications. Typical examples would be tools (compilers, spreadsheets), database systems, user interfaces, etc.

These terms have slightly different meaning in the ODM context. They are not absolute qualities of general functional areas, but describe the span of a given domain's coverage *relative to a particular set of systems*. For example, within a family of large-scale systems, database management may be considered a horizontal domain. However, in the database management systems provider marketplace, database management could be considered a vertical domain. Certain domains like inventory control could have a horizontal or vertical realization in different sets of systems and different business environments.

In ODM, a **vertical domain** refers to a domain scoped to include entire systems in its scope, as shown in Exhibit 30. These systems may be closely related (e.g., a set of system versions or product line) loosely related (a family of systems) or separately developed (i.e., a set of competitive systems addressing a common application area). In vertical domains, reused assets typically form the greater part of the systems created. A **horizontal domain** is a domain that includes only a subset of the functionality implemented in a system. Examples of horizontal domains include database management systems, window managers, and data structures. Horizontal domains are typically smaller in scope than vertical domains and more adaptable to use in many contexts.

The conventional view of vertical and horizontal domains presumes that, for a given system, the lower-level "infrastructure" functions are inherently horizontal, the higher-level, system-specific features inherently vertical in nature. In contrast, following the ODM approach, it is possible to define the boundary of a domain so that it addresses a small portion of the overall functionality for an entire system, but still spans a range of variability determined by coverage of multiple systems. (These are sometimes termed "subsystem-level" domains.) Two independent scoping criteria need to be applied: the number of systems spanned by the domain, and the size of the "system slice" addressed by the domain boundary.

Domains   **Systems**

VERTICAL

**Domain defining a
specified family of systems**

HORIZONTAL

**Domain defining a
functional area spanning
multiple systems/system families**
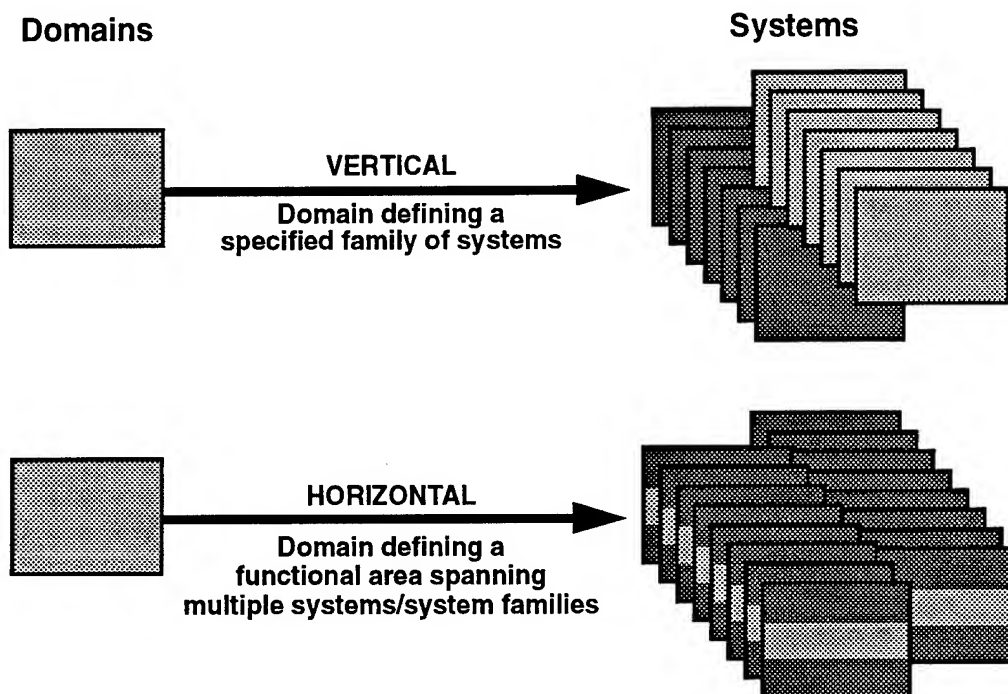
Exhibit 34. Two Kinds of Domains

The conventional view of vertical/horizontal domains encourages a view that equates "applica-tion-specific" (targeted to specific range of application settings) with "vertical." This "family of systems" approach relies on strong assumptions—e.g., that the architectures of the various sys-tems will be fairly compatible, and, more important, that it would be feasible to introduce reus-able assets based on a system-wide "domain architecture." The ODM process can easily accommodate this scenario, but it does not assume it. Moreover, thorough stakeholder analysis prior to the *Scope Domain* sub-phase will usually reveal many strategic barriers to domains of this breadth being successfully transitioned into use within the organization.

Support for Finely Scoped Domains

There are important benefits in considering more tightly scoped domains. A primary benefit is that it is possible to control resources expended on the project to be in line with appropriate investment relative to the size of an average system effort for the organization. Also, it is possible to take an incremental approach to the gradual reengineering of legacy systems. Where it might not be feasible to migrate a new architecture as a whole into a maintained system, it may be possi-ble to migrate reengineered reusable assets on a sub-system by sub-system basis. Sets of systems with diverse architectures can be reengineered to use common sub-system components.

> *Example.* The STARS Army Demonstration project used the ODM concept of vertical vs. horizontal domains defined relative to a specific set of systems in defining their Emitter Loca-tion Processing and Analysis (ELPA) domain. This domain covered a very narrow range of functionality relative to overall system functions, but a heavily algorithmic functional area with significant complexity. While at first picking a domain of limited size (particularly in terms of lines of code) excited some skepticism, the "picket-fence" view of attacking large system maintenance through iterative lateral domain efforts eventually provided a compel-ling story to managers.

Domains   **Systems**

ENCAPSULATED

**Domain defining a
functional area of a system
encapsulated as a separate subsystem**

DIFFUSED

**Domain defining a
functional area distributed
through separate subsystems of the system**
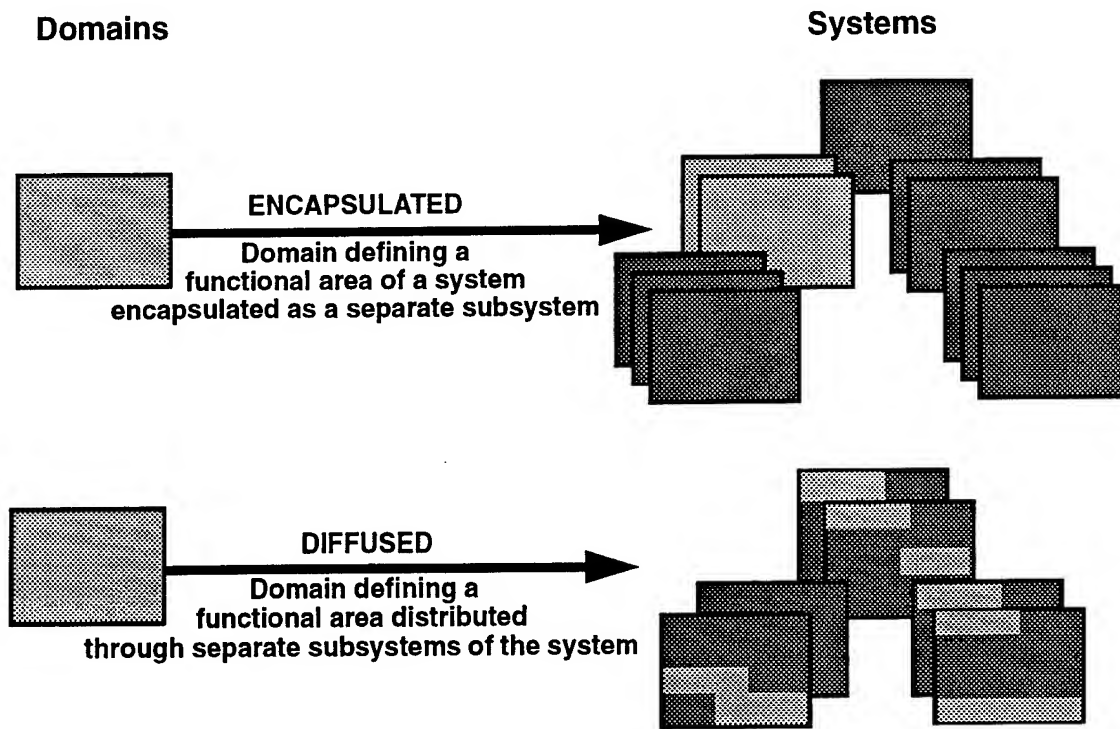
Exhibit 35. Two Kinds of Horizontal Domains

Encapsulated Vs. Diffused Domains

Since any portion of a system can, in principle, be addressed within the boundary of a "horizontal" slice domain, it is in principle possible to target entire applications in an incremental fashion by carving them up into a set of small domains that are targeted with separate (though hopefully coordinated) efforts. The structure of subsystem domains does not necessarily have to match the current architectural decomposition of the legacy systems.

A horizontal domain can be either *encapsulated* or *diffused* with respect to a given system or set of systems, as shown in Exhibit 35. If domain functionality is clustered in one structural component of the system in question, the domain is called an *encapsulated* domain. Conversely a *diffused* domain groups functionality that is pervasive or global within a system.

The advantages of selecting an encapsulated domain are pretty obvious. Because domain functionality corresponds to existing structural boundaries the ability to migrate domain assets into either legacy systems or newly developed systems is facilitated. In addition, such domains will be readily recognizable by people (i.e., they will tend to be native rather than innovative domains).

The advantages of diffused domains are less immediately obvious. But reuse of life cycle products and processes besides code often involves diffused domains. For example, security requirements may be scattered throughout a system requirements specification. An integrated view of these requirements could be valuable topic for a pilot effort, although the requirements are not fulfilled by one particular component of the systems under study.

Later, in the *Acquire Domain Information* sub-phase of *Model Domain*, this data will be important in assessing the effort involved in sifting data for a given portion of the domain. Diffused domains

require more intensive and highly focused data elicitation techniques, to avoid missing data or bringing over too much irrelevant detail.

Somewhat related attributes would be *homogenous* vs. *heterogenous* domains. The former would include domains with common and somewhat uniform internal structure; the latter would include domains that have diverse structural elements. The conventional domain view favors large-scale domains with common architectures. Associated with this approach is the notion that the purpose of domain modeling is to capture only the commonality within the domain. Domains with diverse structure, i.e., heterogenous domains, generally are not handled well by such methods. They are heartily welcomed in ODM, where the goal is very tightly constrained range balanced by as rich diversity as possible within the boundaries.

Results

The primary output of the *Scope Domain* sub-phase is the DOMAIN SELECTION, which has been validated with respect to a set of DOMAIN SELECTION CRITERIA derived from PROJECT OBJEC-TIVES and other strategic data produced in the *Set Objectives* sub-phase. In addition, the DOMAINS OF INTEREST

Process

As shown in Exhibit 33, the *Scope Domain* sub-phase consists of three tasks:

- In the *Characterize Domains of Interest* task, domain engineers perform a scan for possible domains, working with the STAKEHOLDER DOSSIER to ensure that domains important to stakeholders are identified and considered. Direct consideration of PROJECT OBJECTIVES is deferred to the following task to ensure that diverse possibilities for domains within the ORGANIZATION CONTEXT have been probed.

- In the *Define Selection Criteria* task, domain engineers document the DOMAIN SELECTION CRITERIA that will drive the choice of domain, considering general reuse criteria as well as project-specific criteria aligned with the PROJECT OBJECTIVES.

- In the *Select Domain of Focus* task, domain engineers evaluate candidates from the DOMAINS OF INTEREST according to the DOMAIN SELECTION CRITERIA. The output of this task is the DOMAIN SELECTION and refinements to previous workproducts to reflect the domain of focus. These workproducts provide the information for the subsequent sub-phase, *Define Domain*, where the general domain of focus is turned into a formal bounded domain.

Guidelines

- Business area as frame for effort. Separation of the *Characterize Domains of Interest* task works best in situations where there is a clear business area that establishes criteria for completeness of the DOMAINS OF INTEREST list. In situations with weak ties to a coherent line of business (e.g., educational or training settings, or technology evaluation projects where the domain choice is of less significance) there may be less value in separately performing *Characterize Domains of Interest*.

- Use to validate constrained domain choices. *Characterize Domains of Interest* task should not be skipped even if the project appears already constrained in its domain selection. We may need to go back to characterize after defining DOMAIN SELECTION CRITERIA. We may need to reexamine the systems of interest and PROJECT STAKEHOLDER MODEL for viable candidate domains of focus, using DOMAIN SELECTION CRITERIA as a filter.

- <u>Iterate between selection criteria and choosing a domain</u>. Definition of selection criteria in the *Define Selection Criteria* task and selection of a domain in the *Select Domain of Focus* task may need to be performed iteratively. In the *Select Domain of Focus* task, the domain of focus is identified based on the DOMAIN SELECTION CRITERIA. The resultant domain of focus may feel intuitively wrong, fail to get consensus from the project team, or fail to get buy-in and commitment from key stakeholders. Such breakdowns can reveal additional selection criteria and/or different priorities of current selection criteria. Perform the *Define Selection Criteria* task again to account for new selection criteria or weights; then revisit *Select Domain of Focus* based on revised DOMAIN SELECTION CRITERIA.

5.2.1 Characterize Domains of Interest

In beginning the first task of the *Scope Domain* sub-phase, we assume that we have built a picture of the *potential* stakeholders for the project in the STAKEHOLDER DOSSIER. We do not make direct use of the PROJECT OBJECTIVES in this task, in order to encourage exploration of the broader ORGANIZATION CONTEXT from a domain-oriented perspective. Hence, in principle, this task could be initiated before PROJECT OBJECTIVES are complete.
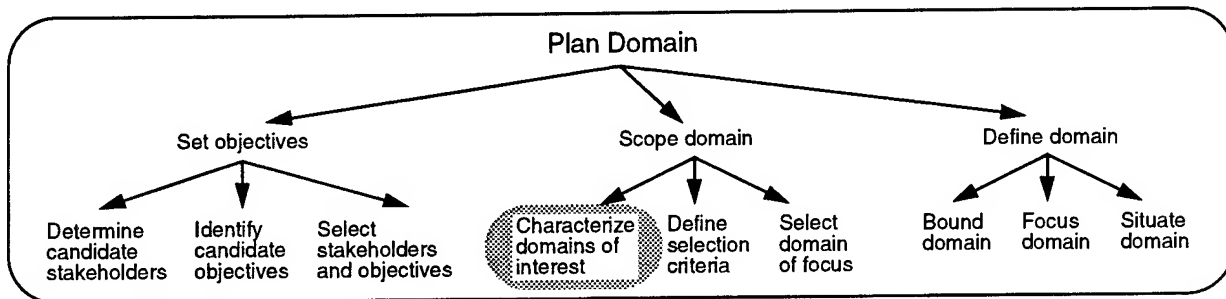
Exhibit 36. Characterize Domains of Interest Process Tree

The primary purpose of the *Characterize Domains of Interest* task is to perform a domain-oriented scan of the ORGANIZATION CONTEXT for the project. Identifying a set of DOMAINS OF INTEREST before selecting a focus domain encourages exploration of alternative domains that may not agree with informal beliefs about reuse within the stakeholder organizations, but may be the best opportunities for reuse within those organizations. Exploration of these alternative domains helps to ensure selection of an optimal rather than merely an acceptable domain of focus.

A major challenge in this task is to balance between spending time identifying familiar domains and spending time identifying innovative domains. Each organization has its own method of naming and organizing functional areas that span systems. The *Characterize Domains of Interest* task allows identification of domains that are named functional areas recognized and familiar to stakeholders. At the same time, the task allows for innovative discovery of domains that are not named functional areas.

Approach

The main function is to identify and discover DOMAINS OF INTEREST within identified systems of interest. Many DA methods are biased towards vertical domains that are decomposed into multiple, encapsulated horizontal domains. This is a best of all possible worlds scenario. In such a domain you can often define a generic architecture w/ plug and play components. But such domains are often not available in many environments. Also, such an approach misses critical areas of variability. Each large system is composed of functionality from many domains. A

generic architecture approach naturally must "lop off" these architecture variants. Similarly, each plug and play component is conceivably usable far beyond the context of the generic architecture in which it was defined. ODM has been designed to systematically reveal a broader set of options in domain identification.

Every system reflects a number of semi-independent design decisions and interfaces. Some of these can be well-encapsulated while others are diffused. There is no way to implement a system where you get the efficiency of optimized diffusion and the maintainability/adaptability of optimum encapsulation simultaneously. Sometimes different systems make different choices along these lines (hybrid domains). Also, over time encapsulated boundaries tend to diffuse (architectural erosion).

> *Example.* Consider the OSI protocol layer model in communications. Ideally an implementation that preserved all the layers would be best from an engineering design standpoint, but it would be unacceptable in terms of performance. So various "optimizations" are applied. Using certain generative techniques (e.g., GenVoca/DAGAR) the advantages of encapsulation may be able to be preserved at design time and optimized away as needed at run-time.

A *domain of interest* is a provisionally named and defined domain that can be linked to stakeholders in the PROJECT STAKEHOLDER MODEL and/or systems of interest to these stakeholders. It is helpful to identify domains of interest as a first step in preparing to select a *domain of focus* for the project, the domain that will be modeled and used as a basis for engineering assets.

However, domains of interest can include more than just domains that are viable candidates for the domain of focus. Central to the approach here is a "restructuring" of the systems of interest in terms of various domain partitionings. These should include whichever of the domain principles for organization are appropriate as discussed in the Approach section of the *Scope Domain* subphase:

- Native domains (matching terms familiar to stakeholders) and innovative domains;

- Vertical and horizontal domains;

- Encapsulated and diffused domains;

- Homogenous domains (those with common architectural principles) and heterogenous domains;

- Domains formed around different products and processes of the system life cycle (e.g., domains focused on requirements, design issues, test data, etc.)

The object is not to decompose systems into some exhaustive list of domains. In fact, understanding domains as boundary decisions shared by groups of stakeholders, it should be clear that there is no way to derive "all the domains" within a given system; at best, this represents the consistent application of one of the "domain-forming" rules mentioned above until the level of granularity can be no longer usefully sub-divided. The object is rather to use various domain concepts as bridges that link organizational and system entities in new ways.

A system is an *exemplar* of a domain if the domain functionality occurs within the system scope. The main approach to characterizing domains in this task will involve mapping which domains intersect which systems, and, similarly, which domains are of interest to particular stakeholder organizations directly. Organizations, systems and domains thus form a three-way set of links that can be explored and cross-validated.

Workproducts

■ DOMAINS OF INTEREST

The DOMAINS OF INTEREST workproduct includes:

- A list of domains, annotated with attributes for each domain, with definitions and values for each attribute.

- A many-to-many mapping of the intersection of DOMAINS OF INTEREST with systems of interest. This mapping characterizes domains by their exemplar systems.

- A many-to-many mapping of DOMAINS OF INTEREST to stakeholders with interests in systems that intersect with the domains.

Exhibit C-5, DOMAINS/SYSTEMS MATRIX, shows a worksheet template that can be used to map the intersection of DOMAINS OF INTEREST with systems of interest. A similar format can be used to map the intersection of DOMAINS OF INTEREST with candidate project stakeholders.

Purposes of the DOMAINS OF INTEREST workproduct include:

- Candidate domains in the DOMAINS OF INTEREST are input to selecting the domain of focus.

- Input to the *Define Domain* sub-phase, used in developing the DOMAIN DEFINITION.

- Reused in subsequent domain engineering projects as a starting point for domain selection. If the ORGANIZATION CONTEXT is stable, the domains of interest to the organization should change relatively slowly (e.g., only if workers develop expertise in new areas, or if existing areas are partitioned into new domains).

When to Start

- Dossier is partially complete. The STAKEHOLDER DOSSIER need not be complete, but should include an initial set of stakeholders, with enough information to identify systems of interest ·(e.g., role of software in the organization).

- Completed objectives are *not* required. The task can be initiated before PROJECT OBJECTIVES have been selected. However, you risk including too many DOMAINS OF INTEREST, since definitive project stakeholders are not determined until objectives have been chosen.

Inputs

- STAKEHOLDER DOSSIER. Used to elicit names of systems of interest to candidate project stakeholders. Domains of interest are scoped to include only domains that intersect these systems of interest.

  This input is not confined to *project* stakeholders. Since DOMAINS OF INTEREST include more than just candidates for the project's domain of focus, the intent of this task is to scan the full range of candidate project stakeholder interests.

- ORGANIZATION INFORMATION. Used to elicit names used within the stakeholder community for various domains. This information may be obtained from:

  — literature which discusses or classifies systems of interest to candidate stakeholders (e.g.,

survey articles or tutorials by acknowledged experts) or

— direct interviews with informants.

- REUSE ATTRIBUTES. General domain-independent attributes that characterize domains that are promising for reuse. REUSE ATTRIBUTES are considered for capture for domains of interest. The domain of focus is later chosen based on values of these and project-specific domain attributes.

Controls

- ORGANIZATION CONTEXT. Used to filter identified domains. May include contextual factors beyond those directly relevant to the PROJECT CHARTER; hence, this control is not intended to limit DOMAINS OF INTEREST to candidate domains of focus. Aspects of ORGANIZATION CONTEXT may also be incorporated as data in workproducts, identifying the organization scope of different domains and systems of interest.

Activities

➤ Identify systems of interest to stakeholders

For each stakeholder in the STAKEHOLDER DOSSIER, identify all software-based systems that are of interest to the stakeholder. Document these systems of interest as a simple annotated list for use throughout this task.

This will involve working from the profiles of the role of software in the organization in the STAKEHOLDER DOSSIER. For example, for a large company looking for reuse within its internal operations systems, the systems of interest will be focused on internal systems. For a product developer, product lines and also competitor's systems available in the marketplace (for which only requirements and user manuals may be available as data) might be systems of interest.

Example. In the Persona scenario, systems of interest include the following:

— PCLaw - developed in cooperation with PCGate and JSoft, with UI technology provided by Persona. PCLaw is under development; decisions are still being made about how it will function.

— Talon - developed by MMI to monitor ER and IC wards. Current version runs with extant hardware and driver software, but in the future, we expect COTS controllers for some monitors, with which Talon's successor must interface. Future versions should be more consistent in 'look and feel' with familiar database programs running in the hospitals using MSWindows.

Other systems are available as either 'best of breed' or 'most popular' for various functionalities.

➤ Record native definitions for the term "domain"

Before generating the list of DOMAINS OF INTEREST, it is helpful to capture stakeholders' definitions of the term "domain". Also elicit any terms besides "domain" that stakeholders use for classes or families of systems and functional subsystems.

The purpose of this activity is for domain engineers to understand stakeholder terminology, in order to identify domains of interest. Stakeholders may have other terms for domains and use the term "domain" differently from how it is used in ODM. Domain engineers may have difficulty identifying domains of interest to stakeholders if the stakeholder terminology is not known. The stakeholders' definitions of the term "domain" allows domain engineers understand what types of domains (i.e., vertical, horizontal, encapsulated, or diffused) stakeholders are talking about when they use the term "domain".

Using interviews and written ORGANIZATION INFORMATION sources determine:

- How do stakeholders differentiate between a domain and a system?

- What kinds of functional areas that span systems are termed domains by stakeholders (e.g., technical areas not bounded by a single contract, project or product)?

If stakeholders have a narrow view of the term "domain", the domain engineers may also want to educate them about other types of domains, so that stakeholders and domain engineers have a common understanding of the term. This would allow stakeholders to help domain engineers identify non-traditional domains that span their systems of interest.

➤ Establish criteria for identifying domains

Domain engineers may filter identified domains before adding them to the DOMAINS OF INTEREST, by specifying criteria for inclusion. DOMAINS OF INTEREST may be constrained to ensure some uniformity in terms of size, complexity, or other measures. This will simplified identification of DOMAINS OF INTEREST because domains will be more consistent in nature. The risk involved with this approach is that novel domains may be excluded. At its simplest, this activity can involve deciding to focus on particular domain characteristics in the identification. For example, a separate "descriptive" pass could be made focusing entirely on the native domains. This would produce a document quite useful in its own right, a sort of "roadmap" of the functional areas recognized within the organization.

Note, however, that these criteria are *not* criteria to filter domains suitable for domain engineering. This is the purpose of the *Define Selection Criteria* task.

➤ Identify domains of interest

This is the key activity of the task. Working from the STAKEHOLDER DOSSIER for organizations, and from the identified systems of interest, identify DOMAINS OF INTEREST by applying the various principles described above. Record each identified domain with a brief explanation in the DOMAINS OF INTEREST.

DOMAINS OF INTEREST can be identified by two methods:

- Based on stakeholders' definitions of the term "domain", elicit examples of specific domains that intersect with systems of interest to the stakeholders.

- Use alternative methods to identify innovative domains that may not be recognized as such by stakeholders. For example, stakeholders may assume that domains must be centered around executable code rather than requirements specifications, or that firmware would not qualify as a software domain.

  These methods include considering unions and intersections of domains derived from stakeholder terms.

In interviewing a system developer, ask about the broad functional areas within a system in their experience. Notice whether they respond in terms of clusters of requirements (security, fault tolerance, error-handling), in terms of structural components of a high-level system architecture, in terms of key issues to be dealt with (application sizing, porting requirements) or the processes supported in terms of work practices. Specifically ask practitioners about areas where they have observed a pattern of recurring commonality across systems. Look for artifacts like survey articles that provide "roadmaps" of the functional area.

In using alternative methods for deriving innovative domains, use concepts presented in this guidebook (e.g., of diffused domains) to search for ways of grouping functionality that may not have occurred to practitioners.

> *Example.* In the Persona scenario, a meeting is held with the team to generate the DOMAINS OF INTEREST. Many of the categories come from established "niches" in PC software, such as window managers, database displays, spreadsheets, etc. The domain of "UIs for medical records exchange" excites some discussion because it combines what people think of as a general-purpose horizontal domain (UIs) with a vertical "niche" (medical records exchange). Some discussion ensues about records exchange problems in non-medical areas (police reports, etc.) and presumed commonalities and variabilities.

> Returning back to the systems of interest, and checking the criteria for domain identification, the group decides to make an effort to discover some innovative domains. Two particular systems of interest, PCLaw and Talon, are investigated. "Can we identify any apparent areas of overlapping functionality required in the UI components for these two systems?" Engineers in the meeting note that a requirement for viewing of heavily nested hierarchical data in outline form is a common requirement of both systems.

> Some skepticism is raised, because these functions are not called out as separate functional components, or even specified via well-encapsulated requirements. However, Jack Perricone points out that "outliner modes" have become standard fare in PC text editors and other tools. They agree to include "Outliners" in the DOMAINS OF INTEREST.

➤ Map domains to systems

Map the intersection of DOMAINS OF INTEREST with systems of interest. Systems of interest were determined in the first activity in this task. For each system of interest determine domains in the DOMAINS OF INTEREST that intersect with that system. Record known relationships between the DOMAINS OF INTEREST and systems of interest, but do not attempt a complete mapping at this time. A complete exhaustive mapping between systems of interest and domains is not needed at this time. A worksheet template that can be used for this activity is shown in Exhibit C-5, DOMAINS/SYSTEMS MATRIX.

An excerpt of a DOMAINS/SYSTEMS MATRIX for the Persona scenario example is shown in Exhibit 37. The Outliner domain turns out to be quite diverse; somewhat encapsulated in one system, diffused in the functionality of another, and existing as a stand-alone PC application as well. This seems to suggest a high degree of variability in architecture that would need to be addressed to handle the domain.

Domains of Interest	Systems of Interest					
	PCLaw	Talon	MS Word	MS Access	Web Arranger	MUMPS
Window managers		E	E	E	E	
Database displays	E	E		E		D
Outliners	E	D	E		V	
UI for legal services	D					
UI for medical records exchange		V		D		E

Exhibit 37. *Example Excerpt:* Domains/Systems Matrix

➤ Map domains to stakeholders

Determine which domains in the DOMAINS OF INTEREST are of interest to which stakeholders. To perform this activity, use the mapping between DOMAINS OF INTEREST and systems of interest from the previous activity, and the list of systems of interest to each stakeholder from the first activity in this task. A domain is of interest to a stakeholder if one of the systems of interest that intersects with the domain is of interest to the stakeholder. A format similar to that used in the DOMAINS/SYSTEMS MATRIX in Exhibit C-5 can be used.

Correlate the DOMAINS OF INTEREST derived from stakeholder views and system views respectively, by completing the other side of the equation: i.e., for a domain derived from the stakeholder view, consider its intersection with systems of interest, and vice versa. This yields a nice cross-hatch of domains as intersections of systems and organizational entities.

➤ Characterize domains

For each of the DOMAINS OF INTEREST, characterize the domain by identifying domain values for each of the attributes chosen in the previous activity. Characterize which domains are vertical, which horizontal/encapsulated, and which horizontal/diffused, etc.

As further characterization, select attributes from general REUSE ATTRIBUTES that are relevant and add attributes that are particular to the ORGANIZATION CONTEXT. For each domain of interest a value will be determined for each of these attributes. Define each selected attribute and its possible values. Values can be numeric, high/medium/low, yes/no, etc. The following starter list contains some general REUSE ATTRIBUTES:

• Maturity of the domain including number of systems implemented and length of time fielded.

• Potential for future payoff. How many systems are to be implemented in the future?

• Stability of underlying technology within the domain. Does the domain encapsulate technical approaches that are due to become obsolete shortly?

- Availability of codified knowledge in the domain.

- Experience in the domain held by the organization performing the domain engineering project. Is this organization a developer of systems in the domain? A maintainer? A user?

- Estimated commonality of functions across systems that intersect with the domain.

- Degree to which performance constraints may inhibit reuse of components with excess functionality.

Also, knowing the intersection of a domain of interest across a set of systems of interest helps us to derive some of these properties; e.g., we can determine properties like the maturity (number of fielded systems) the stability (number of anticipated systems in future) by looking at the distribution of the domain across legacy and anticipated systems of interest.

Note that we are not using reuse attributes as a way of *screening* domains of interest. Some domains of interest will have been worth documenting that have little practical value for reuse efforts. Here we do an initial characterization in terms of reuse as preparation for the *Define Selection Criteria* task. (This could also be deferred until that task.) By associating this information with the DOMAINS OF INTEREST workproduct, we separate out general reuse criteria from project-specific and more tactical criteria, considered in the next task. Thus the DOMAINS OF INTEREST should be a useful starting point for subsequent domain engineering efforts involving the same basic set of stakeholder interests.

When to Stop

- Native domains addressed. All domains named by stakeholders have been included in the DOMAINS OF INTEREST.

- Systems of interest intersect a domain. All systems of interest to candidate project stakeholders are included in at least one vertical domain.

- Domains are mapped to a system of interest. Each domain of interest is mapped to at least one system of interest.

Selection of the domain of focus is *not* an exit criterion; it is performed later in this sub-phase.

Guidelines

- Do not screen out poor reuse choices (yet). When determining DOMAINS OF INTEREST, do not discard domains solely because they appear to be poor candidates for the application of reuse technology. DOMAINS OF INTEREST should not be screened because they are also used in the *Define Domain* task, where they are a source of information about domains related to the domain of focus.

- Do not attempt exhaustive cataloging of domains. Since domains are not merely modules or components of systems, there are no absolute criteria for completeness of DOMAINS OF INTEREST for a given ORGANIZATION CONTEXT. In particular, while each domain should be traceable to *some* stakeholder and *some* system of interest, don't attempt to elicit all such relationships in this step. DOMAINS OF INTEREST selected can be "fractal" in that greater levels of detail and criteria for decomposition emerge the more attention is paid to the domain material. Their boundaries can be overlapping, and infinitely refinable. More complete elicitation of these relationships is performed for the domain of focus in the final task in this sub-phase, *Select Domain of Focus.*

- Apply set transformations. It can be helpful to loosen up thinking about domains to create new DOMAINS OF INTEREST by treating already identified domains as sets and applying simple operations like looking for the union and intersection of two domains, aggregating smaller domains into composites which can then be assessed for certain economies of scope or synergistic benefits through combinations of different functional areas.

5.2.2 Define Selection Criteria

We now have a set of DOMAINS OF INTEREST, characterized with respect to stakeholder organizations and systems of interest. We also know which domains reflect "ethnographic" categories (terms used by domain practitioners) and which reflect innovative ways of "slicing the cake."
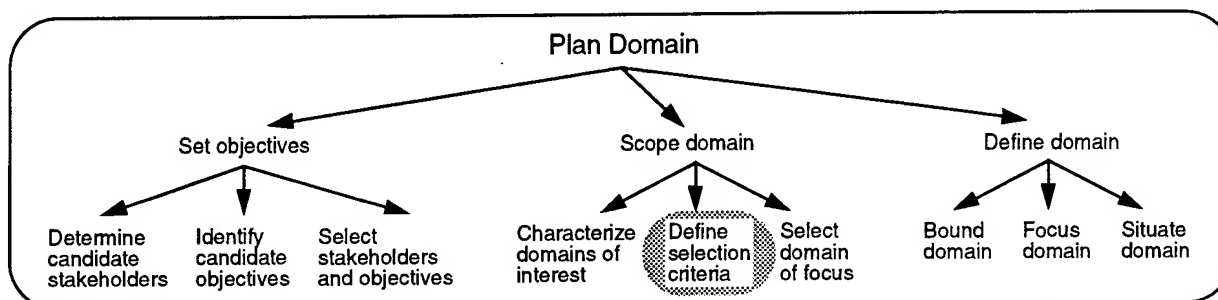
Exhibit 38. Define Selection Criteria Process Tree

The primary purpose of the *Define Selection Criteria* task is to link this information with PROJECT OBJECTIVES, to determine domain selection criteria. These criteria will be used in the *Select Domain of Focus* task to choose a domain of focus from the DOMAINS OF INTEREST. Documenting DOMAIN SELECTION CRITERIA helps to reveal implicit conflicts between organization beliefs and general criteria that identify domains with reuse potential, prior to domain selection. It also provides a means of allowing upper management to sign-off on the criteria used for selecting the domain of focus, while leaving the actual selection task under the control of the domain engineering project members. A major challenge of the task is to identify all the *tactical* criteria that affect the feasibility of a domain for the immediate effort. Often, these criteria will be obscured because political considerations prevent their clear articulation, or they will be described too broadly to serve as an effective aid in scoping a domain of manageable size.

Another challenge in this task is that it is possible to derive conflicting criteria from a consistent set of PROJECT OBJECTIVES. Do we want innovative vs. ethnographic domains? Diffused vs. encapsulated? e.g., diffused could make it harder to reengineer but offer greater payoff if you succeeded. What's the willingness for up-front investment, the tolerance for risk, the planning window within which payoff must be demonstrated?

Another challenge requires a word of caution against too blithe an acceptance of industry "conventional wisdom" about selection criteria for domains. Many of these recommendations embed assumptions about the type of organization doing reuse that may not hold for your situation.

Approach

The approach to the *Define Selection Criteria* task is to determine DOMAIN SELECTION CRITERIA based on general criteria that identify domains with reuse potential, and to integrate and reconcile general reuse criteria with PROJECT OBJECTIVES. A benefit of the separate criteria identification task is enabling this cross-checking of organizational, tactical criteria with general criteria.

Criteria for domain selection are determined separately from the *Characterize Domains of Interest* task because there is value in characterizing domains that would clearly not be good candidates for domain engineering based on the project context. For example, a company involved in data-intensive applications should carefully consider the evolving DBMS domain, although the company may not have such a domain within its own engineering scope.

Criteria for identifying good domains for reuse are also determined separately from domain selection in the *Select Domain of Focus* task. The same criteria can be useful for many iterations of domain selection. These iterations of domain selection can be used to change focus when resources expand or contract. While criteria may include domain attributes such as availability of domain experts, the current life cycle phase of likely recipient application projects, etc., actual PROJECT CONSTRAINTS are addressed in the *Select Domain of Focus* task.

Workproducts

■ DOMAIN SELECTION CRITERIA

A list of prioritized selection criteria that will be used to select the domain of focus, with annotations explaining the derivation of each criterion. Also included is a mapping from domain selection criteria to PROJECT OBJECTIVES to show whether each domain selection criterion supports or conflicts with each objective. Exhibit C-6, PROJECT OBJECTIVES/CRITERIA MATRIX, contains a worksheet template that can be used to map DOMAIN SELECTION CRITERIA to PROJECT OBJECTIVES.

When to Start

- Domain attributes have been selected and defined. Domain attribute names, definitions, and value ranges were determined as part of the DOMAINS OF INTEREST workproduct developed in the previous task. The entire DOMAINS OF INTEREST workproduct does not have to be completed to begin the *Define Selection Criteria* task. As soon as domain attribute descriptions and possible attribute values are available, general reuse criteria can be derived based on domain attributes and their values.

- A domain of interest has been characterized. Selection criteria based on a domain of interest can be derived as soon as the domain is characterized.

Inputs

- DOMAINS OF INTEREST. Domain attributes are one source of DOMAIN SELECTION CRITERIA. The mappings of systems of interest to stakeholders are also used to determine selection criteria based on systems which are of interest to key stakeholders.

  Note that any correlations of general REUSE ATTRIBUTES to domains flows into this task through this input.

Controls

- PROJECT OBJECTIVES. Used as the primary means of filtering selection criteria into those appropriate for the project. Also an input used to determine DOMAIN SELECTION CRITERIA.

Activities

➤ Derive criteria from domain attributes

Choose domain attributes (recorded as part of the DOMAINS OF INTEREST) relevant to the selection of a domain with reuse potential. For each chosen domain attribute, determine the range of attribute values that identify a domain with reuse potential. Record these as DOMAIN SELECTION CRITERIA. Domain selection criteria could include:

- Maturity: A repertoire of existing systems exists for the domain.

- Stability: The domain is stable, so that technology change will not quickly render it obsolete.

- Performance: Performance requirements for the domain are attainable.

➤ Derive criteria from objectives

Derive DOMAIN SELECTION CRITERIA based on attributes of a good domain of focus derived from the PROJECT OBJECTIVES. Each objective may yield several selection criteria. Assemble selection criteria independently for each objective. The mapping of stakeholders to systems of interest (part of the DOMAINS OF INTEREST) can also be used to develop selection criteria that are sensitive to the interests of key stakeholders.

> *Example.* Domain selection criterion based on Project Objectives could include:
>
> — The domain is of strategic interest to stakeholder X.
>
> — Domain experts are available within the organization.
>
> — A domain engineering team can be formed for this domain within the current organization structure.
>
> *Example.* One selection criterion for the Army/Lockheed Martin STARS Demonstration Project [ADER96a] was that the domain selected should cover functionality contained in software on the main computer used by the exemplar systems, rather than firmware-embedded functionality on peripheral equipment. This criterion was a project-specific criterion linked to the PROJECT OBJECTIVES, not a general reuse criterion for domain selection.
>
> As a counter example, for some Hewlett Packard application groups, reuse within firmware domains is entirely appropriate to their business area.

For each selection criterion, determine the range of values that the domain of focus must fall within to satisfy the selection criterion.

> *Example.* A project objective could be to "validate technology for developing flexible asset base architectures." This objective suggests that the degree of structural variability in the architecture of the domain would be a significant domain selection criterion. Degree of structural variability would be added to the criteria list, with a preferred value of greater as opposed to less variability indicated.

➤ Merge selection criteria

DOMAIN SELECTION CRITERIA and satisfying values that were derived from various sources are merged into an annotated list that includes derivation of each criterion. Criteria can conflict and even be mutually exclusive at this point.

➤ <u>Map criteria against objectives</u>

Map all DOMAIN SELECTION CRITERIA back to the complete list of PROJECT OBJECTIVES. This is a characteristic ODM validation step: after deriving data from a particular source, we map the data back across all source data to check for conflicts and potential synergies. In this case, some criteria will have been derived from individual objectives. For each selection criterion, determine which project objectives support or conflict, or are neutral with respect to the criterion. Exhibit C-6, PROJECT OBJECTIVES/CRITERIA MATRIX, contains a worksheet template that can be used for this activity.

> *Example.* For the Persona example, an excerpt for this worksheet is shown in Exhibit 39. Two conflicts are noted: 1) satisfying the objective of providing functionality not well covered by COTS systems appears to conflict with the selection criterion of finding functionality that is highly visible in popular commercial systems. And 2) Any capabilities easily amenable to Web and ORB interfaces may be likely to have already been addressed in a COTS API. (These issues will turn out to have bearing on the final domain selected.) Note that the point of the exercise is certainly not to eliminate these identified conflicts, which will be ever-present. Instead, this provides a framework for thorough tradeoff analysis for each domain considered.

Project Objectives	Domain Selection Criteria										
	Use in Web and CORBA applications	Cuts across platforms	Visible in popular systems	Not in COTS System	When in COTS, not accessible via API	Impacts turnkey systems	Of use in planned systems	No reliance on technology soon obsolete	Knowledge available	Commonality across systems	Little performance impact from extra functionality
Provide capabilities not currently available from COTS APIs	X APIs as candidates for ORB or Web	0	X	+	0	0	+	+	0	0	0

Exhibit 39. *Example Excerpt:* Project Objectives/Criteria Matrix

➤ <u>Prioritize criteria</u>

Derive a final prioritized, annotated list of DOMAIN SELECTION CRITERIA, based on the mapping of selection criteria to PROJECT OBJECTIVES. If the list is a subset of the original list of selection criteria, rationale should be recorded for rejected criteria.

When to Stop

- <u>Criteria are aligned with objectives</u>. Criteria definition is complete when the prioritized list of DOMAIN SELECTION CRITERIA has been generated and alignment of each criterion on the list to PROJECT OBJECTIVES has been demonstrated.

Guidelines

- <u>Criteria should reflect project realities</u>. Criteria should reflect as much as possible the particular project context in which the selection is being made. No criterion is wrong if it reflects a contextual constraint. Do not ignore situation-specific constraints that may seem to obscure the purity of general technical reuse criteria.

- <u>Do not ignore general reuse criteria</u>. Project-specific criteria may override general reuse criteria in some situations. However, general criteria cannot be ignored. For example, if a compelling reason cannot be made for reuse of assets across multiple systems in a given business area, general reuse criteria would say that domain engineering should not be performed on this domain. However, the organization may have a desire to devote resources to software process improvement in this domain.

- <u>Examine domain selection mandates</u>. If the PROJECT OBJECTIVES contain assumptions about which domain is selected for domain engineering, include these assumptions in selection criteria. Other selection criteria may contradict domain selection assumptions. If this is the case, the selection criteria must be resolved. If resolution is impossible, the project may have to terminate. Although termination is undesirable, it is better to terminate than to than pursue domain engineering on a domain which is a poor risk.

5.2.3 Select Domain of Focus

We have now identified domains of interest and selection criteria for selecting the domain. The primary purpose of the *Select Domain of Focus* task is to select a domain of focus for the domain engineering project that satisfies PROJECT OBJECTIVES and is feasible given PROJECT CONSTRAINTS, including project resources.
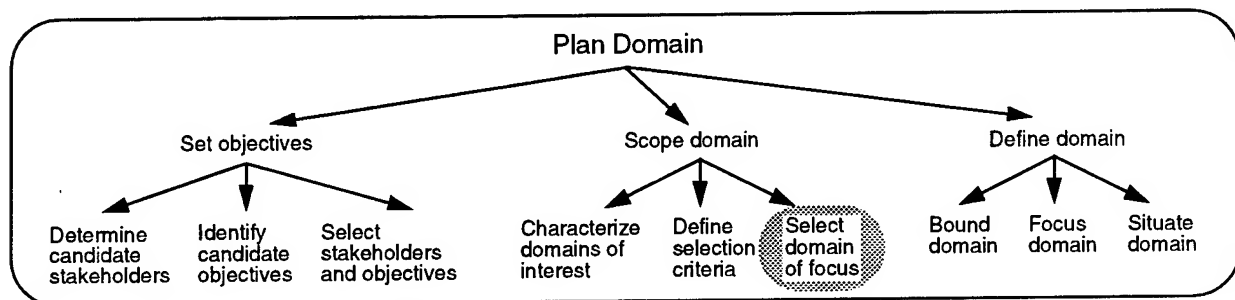
Exhibit 40. Select Domain of Focus Process Tree

The main challenge of this task is, of course, picking the right domain! This decision has the greatest single impact on a successful domain engineering project outcome. Poor domain modeling using the right domain can be repaired with rework. The best modeling expertise in the world, applied to a domain that is a poor strategic choice, will have little chance of success.

But "selection" here involves several elements, including scoping (determining how wide or narrow to draw the boundaries of a domain centered on a given key area), risk mitigation based on alignment of the domain to identified objectives, and, as with the *Select Stakeholders and Objectives* task, obtaining solid commitment from stakeholders.

Approach

The main approach to the *Select Domain of Focus* task is to make a DOMAIN SELECTION from the DOMAINS OF INTEREST, using the DOMAIN SELECTION CRITERIA. As a subsidiary function, various workproducts are created or updated to reflect the commitment to a specific domain. These include:

- The PROJECT OBJECTIVES are augmented with domain-specific objectives,

- CANDIDATE EXEMPLAR SYSTEMS for the selected domain are extracted from the systems of interest.

- The DOMAIN STAKEHOLDER MODEL is extracted from the PROJECT STAKEHOLDER MODEL by choosing stakeholders who are interested in the selected domain.

Workproducts

■ DOMAIN SELECTION

Documents the domain that was selected as a domain of focus, and the rationale for choosing this domain. The DOMAIN SELECTION includes:

- A characterization of candidate domains in terms of the DOMAIN SELECTION CRITERIA. For each candidate domain, this characterization tells whether the domain satisfies or conflicts with each selection criteria.

- Include the various characterized attributes of the selected domain: native/innovative, vertical/horizontal, encapsulated/diffused, etc.

- The name of the selected domain of focus, along with the rationale for the selection.

- An informal description of the selected domain of focus.

Exhibit C-7, DOMAINS/CRITERIA MATRIX, contains a worksheet template that can be used to evaluate each domain of interest with respect to the identified DOMAIN SELECTION CRITERIA.

■ CANDIDATE EXEMPLAR SYSTEMS

Indicates systems of interest that intersect the domain of focus, characterized with respect to this intersection. This list will be further refined in the *Define Domain* sub-phase, where the domain boundaries are fully defined.

■ PROJECT OBJECTIVES

PROJECT OBJECTIVES are augmented with specific objectives for the project related to the domain of focus. Also identifies potential scenarios for reuse, based on the chosen domain and project objectives.

■ DOMAIN STAKEHOLDER MODEL

A refinement of the PROJECT STAKEHOLDER MODEL which only includes stakeholders with an interest in the domain of focus. The DOMAIN STAKEHOLDER MODEL includes any new stakeholders with interest in the domain, based on the informal domain description. The DOMAIN STAKE-

HOLDER MODEL also includes the specific roles of all stakeholders with respect to the domain of focus.

When to Start

- Some DOMAINS OF INTEREST have been identified. It is possible to begin identifying candidates for the domain of focus as soon as some DOMAINS OF INTEREST have been identified in the *Characterize Domains of Interest* task.

- Selection criteria are available for detailed trade-off analysis. Detailed trade-off analysis should not begin until DOMAIN SELECTION CRITERIA have been finalized.

Inputs

- PROJECT OBJECTIVES. Augmented with objectives specific to the domain of focus. PROJECT OBJECTIVES are *not* a control on this task. The DOMAIN SELECTION CRITERIA should have incorporated any PROJECT OBJECTIVES relevant to the domain selection task.

- DOMAINS OF INTEREST. Used as the source for candidates for the domain of focus. Also, the mapping of domains to systems of interest is used to determine the strategic value of candidate domains under consideration, in terms whether they intersect systems of interest to key stakeholders.

- PROJECT STAKEHOLDER MODEL. A source of stakeholders with interest in the selected domain.

Controls

- DOMAIN SELECTION CRITERIA. Control the selection of the domain of focus.

- PROJECT CONSTRAINTS. Some constraints have already been factored into PROJECT OBJECTIVES and the PROJECT STAKEHOLDER MODEL. In selecting the domain of focus more detailed constraints become relevant. For example, project resources, schedules, available .personnel, etc. must be considered carefully in choosing a feasible domain scope.

Activities

➤ Document candidate domains of focus

Choose domains from the DOMAINS OF INTEREST that are candidates for the domain of focus. Some DOMAINS OF INTEREST may not be candidates. For example, if the project is restricted to choosing a domain of focus within a certain line of business, then only DOMAINS OF INTEREST within the line of business are candidates for the domain of focus.

➤ Evaluate candidate domains

Evaluate each domain of interest with respect to the identified DOMAIN SELECTION CRITERIA. Based on the priority assigned to each of the DOMAIN SELECTION CRITERIA and the criteria values recorded for each candidate domain, produce an ordered list of candidates. Exhibit C-7, DOMAINS/CRITERIA MATRIX, contains a worksheet template that can be used for this activity.

> *Example.* For the Persona scenario, an excerpt from an example DOMAINS/CRITERIA MATRIX is shown in Exhibit 41. By an uncanny coincidence, the analysis suggests that the Outliner

domain is the strongest candidate domain. We will point out just one insight apparent from the data in the worksheet.

The objective of finding a domain that has functionality familiar from COTS systems, but that is not accessible via standard APIs from such systems, is a major strategic angle for Persona. Finding such a domain and producing best-of-breed functionality and quality of service they can differentiate their offering in the highly competitive and feature-driven UI marketplace. This objective has been translated into the domain selection criterion, "When in COTS, not accessible via API." The Outliner domain is the only one considered that matches this criterion. Outliner "modes" are familiar from COTS products, but it is not easy to provide this functionality within turnkey systems by embedding COTS applications, as would be the case with, say, a spreadsheet program.

Candidate Domains of Focus	Domain Selection Criteria (Weighted)									Total
	Use in Web and CORBA applications	Cuts across platforms	Visible in popular systems	When in COTS, not accessible via API	Impacts turnkey systems	Knowledge available	Experience held by persona	Commonality across systems	Little performance impact from extra functionality	
Weight	1	1	1	2	1	2	2	1	1	
Window managers	−	−	−	0	−	+	+	+	−	+0
Database displays	+	+	+	−	+	+	+	+	0	+7
Looks and Feels	−	+	+	−	−	+	+	+	−	+2
Outliners	+	+	+	+	+	+	+	+	+	+12
UI for legal services	0	+	−	0	+	0	−	0	+	+0
UI for medical records exchange	0	+	−	0	+	0	−	0	+	+0
MOTIF Widgets	−	+	+	−	0	+	+	+	−	+3

Exhibit 41. *Example Excerpt:* Domains/Criteria Matrix

➤ Factor in other considerations

Examine the prioritized list of candidate domains and note other benefits and risks of selecting particular domains. These will be factors unique to a specific domain that were not listed in the

DOMAIN SELECTION CRITERIA. For example, the selection of a certain domain might present an opportunity for one of the domain asset users to be a new development project.

The DOMAIN SELECTION CRITERIA should produce a list of viable candidate domains. Project resources and other PROJECT CONSTRAINTS are now used to filter the list of candidate domains into a list of those that can be engineered within PROJECT CONSTRAINTS.

> *Example.* In the Persona scenario, the OMI Inc. consultant has cleverly directed the conversation towards the Outliner domain. He now points out that he has material available from a previous domain modeling effort on commercial outliner capabilities in PC and Mac based applications that can be applied to the pilot effort. This provides additional incentive for the domain choice. This also gives the consultant a chance to participate more directly in the modeling process itself.

➤ Select the domain of focus

Using all the information obtained in the preceding activities, make the final DOMAIN SELECTION.

Obtain buy-in from the entire project team on the domain selected. Not all domain engineering decisions can be made on a collaborative, consensus basis. However, the choice of domain has pervasive impact on the entire project. Any disaffected parties with regard to this decision may cause difficulties later in the project.

In the DOMAIN SELECTION workproduct, record an informal description of the domain of focus and the rationale for this selection. Also document advantages and risks of performing domain engineering on this domain of focus.

➤ Identify and characterize domain stakeholders

The PROJECT STAKEHOLDER MODEL includes people and organizations recognized as having some stake in the domain engineering project as a whole. Now that the domain of focus is selected, we refine this model into the DOMAIN STAKEHOLDER MODEL, a model of stakeholders specific to the domain. Not all project stakeholders will have a stake in the results of the project now that a specific domain of focus has been chosen.

Like other refinement activities in ODM, PROJECT STAKEHOLDER MODEL refinement may involve a combination of adding, changing, and removing information. Refinement of the PROJECT STAKEHOLDER MODEL into the DOMAIN STAKEHOLDER MODEL involves the following transformations:

- Include project stakeholders who are interested in the domain of focus in the DOMAIN STAKE-HOLDER MODEL.The mapping of domains to stakeholders in the DOMAINS OF INTEREST workproduct can be used here.

- Remove project stakeholders from the PROJECT STAKEHOLDER MODEL whose stake was contingent on the selection of a particular domain (or type of domain) that is not the domain of focus. The domain selected does not match their interests. Not only will these stakeholders not show up in the DOMAIN STAKEHOLDER MODEL, but their role in the PROJECT STAKE-HOLDER MODEL may be radically changed. It may even be appropriate to remove them from the PROJECT STAKEHOLDER MODEL if they are no longer interested in the project.

  If a project stakeholder who is not interested in the domain of focus is a *key* project stakeholder, such as one of the project funding entities, this is a risk factor that should be looked at closely. Has the stakeholder lost a primary incentive for involvement? Might issues arise

downstream if the stakeholder does not participate?

- Decide not to include project stakeholders in the DOMAIN STAKEHOLDER MODEL if their stake is not contingent on the domain of focus, but who have no particular interest in the domain selected. The nature of the relationship of these stakeholders to the project is not changed by the selection of the domain. These stakeholders will remain "active" members as reflected in the PROJECT STAKEHOLDER MODEL but will not be included in the DOMAIN STAKEHOLDER MODEL.

  For example, a research director may have decided to fund a domain engineering effort as a trial effort to see whether the approach will make sense for the corporation. The research director may not be directly interested in the particular domain selected provided it meets certain overall criteria. As another example, a technology provider to the project may be a stakeholder because of the provider-customer relationship. This provider may also have no particular stake in the domain selected.

- Document any changed or more specific interests of stakeholders with respect to the selected domain.

- Add any new stakeholders suggested by the nature of the domain. Review the rationale for domain selection and ask whether the rationale suggests any new stakeholders that need to be considered.

Stakeholders in the DOMAIN STAKEHOLDER MODEL should be characterized as to the roles they may perform with respect to the project. These roles are all tentative at this point, but it is helpful to consider each stakeholder from these various perspectives and to make sure that some stakeholder is identified for all or most of the roles. Roles without an identified stakeholder are potentially an area of risk (unless, for example, no access providers are listed because information for the domain is readily available). Roles can include:

- Project team participants. Are there be new people available to participate in domain modeling or asset base engineering, given the domain selected?

- Sources of domain knowledge. Note stakeholders who may be good sources of information about the domain of focus. Was expectation that this person will be available to share domain knowledge a deciding or strongly influencing factor in selecting the domain? In some cases there may be this strong dependency on the availability of specific personnel. Such personnel will often become not only sources of domain knowledge but also advisors about other people with domain knowledge as well.

- Access providers. Is this stakeholder essential in order to obtain access to artifacts or people who will provide domain information?

- Potential users of workproducts produced by the project, including the DOMAIN DEFINITION, DOMAIN DOSSIER, DOMAIN MODEL, ASSET BASE, or ASSETS. The typical user for assets will be an application developer building a software product.

➤ Update and refine objectives

The general PROJECT OBJECTIVES are refined into more specific statements that involve desired project results in the selected domain. Other domain-specific objectives are added to the PROJECT OBJECTIVES.

Example. One of our PROJECT OBJECTIVES was "The example domain engineering effort should effectively demonstrate the utility of subsystem level domains, since these are less

familiar to many software engineers than larger, whole-system domains." This became one criterion for domain selection. Having selected the Outliner domain, this original objective now translates to the domain-specific project objective "Show examples of how Outliner domain functionality occurs as a sub-system within several familiar application contexts (e.g., text editing, file-system browsing, and presentation management.)"

There are several other possible sources of domain-specific PROJECT OBJECTIVES. DOMAIN SELECTION CRITERIA satisfied by the selected domain can be used as a source of these objectives. The documentation of the roles and interests of domain stakeholders in the DOMAIN STAKE-HOLDER MODEL can also be a source of domain-specific objectives.

➤ Refine systems of interest into CANDIDATE EXEMPLAR SYSTEMS

Based on the informal domain description in the DOMAIN SELECTION, select CANDIDATE EXEMPLAR SYSTEMS for the domain from the systems of interest. This information can be largely derived from the mapping of domains to systems of interest within the DOMAINS OF INTEREST workproduct. Candidates are those systems of interest which have a significant intersection with the selected domain. New systems might have been considered as well.

➤ Obtain commitment from stakeholders

Necessary commitment from stakeholders includes:

- Permission to proceed with domain engineering.

- Commitment of resources necessary to perform domain engineering on the selected domain.

- Backing to carry out the project. This includes permission to:

  — cross certain organization boundaries (e.g., talk with customers and system developers),

  — talk to people with expertise in the domain,

  — access documentation, etc.

When to Stop

- Essential selection criteria area satisfied. The selected domain of focus satisfies all essential DOMAIN SELECTION CRITERIA.

- Selection rationale documented. Rationale for the DOMAIN SELECTION has been documented. This documentation is especially important if the DOMAIN SELECTION CRITERIA priorities do not fully motivate the choice.

- Project team has consensus. There is consensus among the project team for the choice. The selection of a domain of focus is one task where a team-based approach is essential to project success.

It is also possible that the project will terminate at this point if:

- No domain satisfies the criteria. If no domain can be found which satisfies the essential DOMAIN SELECTION CRITERIA, iterate back to *Characterize Domains of Interest* to search for more domains. If a few iterations do not resolve the impasse, consider stopping the project.

NOTE: At various places in the process model we give advice to consider termination of the

project. By and large, especially early in the life cycle, such termination conditions are a sign that the process is doing its job, rather than allowing at-risk projects to lumber forward, lose momentum and consume precious resources.

Guidelines

- Multiple domains chosen. The ODM process model assumes the context of a single domain. If multiple domains are chosen for engineering, determine the relationships between the domains. If the domains form a cohesive cluster, then the set of domains form the domain of focus for the project. If the domains are not related, then domain engineering should be performed on each domain separately.

- Document the real rationale. It is sometimes the case that priorities documented in the DOMAIN SELECTION CRITERIA do not capture some key factors used in choosing the domain of focus. After listing criteria and making choices, the "real but non-rational" reasons for the final choice (such as political reasons) should become more visible. The challenge is then remembering (or being willing) to capture these other rationale, once they are discovered. The fact that certain criteria are initially invisible is, in itself, powerful data about the strategic issues involved.

  Document the real concerns and rationale that motivated the DOMAIN SELECTION as much as possible. Judge the overall domain selection process, not by asking "Was our original list of criteria complete?" but rather "Did the process help us become more explicit about decision criteria that might otherwise have been assumed and unexamined?" Documentation of the real rationale for domain selection allows the project to attempt to mitigate any risk caused by this selection.

5.3 Define Domain

Up to now, the *Plan Domain* phase has been a "narrowing" process: from the interests of various stakeholders to a specific set of project objectives, to a set of candidate domains, narrowed further in the *Select Domain of Focus* task to a domain of focus for the project. The process so far has worked with an informal, intuitive understanding of the domain of focus. Care has been taken to ensure that the selected domain makes strategic sense for the business context of the domain engineering project.
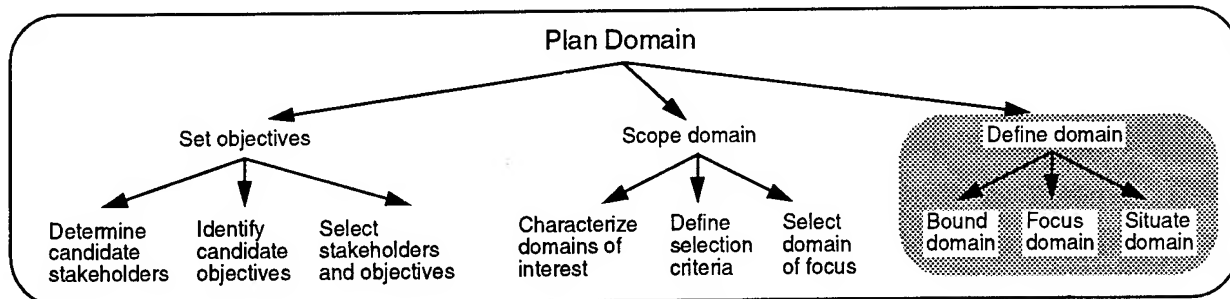
Exhibit 42. Define Domain Process Tree

The goal of *Define Domain* is to establish clearer and more formal boundaries around the selected domain, boundaries that respect both strategic and conceptual criteria. A complementary goal of this task is to widen planners' perspectives to consider the domain from multiple viewpoints. This serves as an overall validation of the selection process, helps anticipates later modeling issues up front, and can aid in coordinating the project with other efforts.

There are several challenges involved in the definition process. Definition of boundaries for complex entities like software systems, or portions of software systems, is not a simple undertaking. Definition of shared conceptual entities like domains is even more difficult. We think we know what our domain is after we've selected it. This sub-phase is our chance to find out the many ways in which we do *not* know what the domain is.

- *Strategic vs. Formal Definitions.* There is a fundamental trade-off between strategic and formal criteria for domain definition. The shift to *Define Domain* is a qualitative transition where stakeholder issues, though still present and important, are balanced with conceptual concerns. While choice of domain can be a decision dominated by business processes, finding intuitively compelling boundaries for the chosen domain is not just a stakeholder-driven decision. After we strategically choose a central area of focus for a domain, we encounter various constraints in drawing intuitive and stable boundaries for the domain. These constraints reflect a host of other criteria for the cohesiveness and distinctiveness of the functionality included in the domain: technical, functional, or even appealing to certain aesthetic preferences of the domain engineers.

  A definition process must somehow integrate these strategic and "formal/conceptual" principles. A definition based entirely on perceived stakeholder interests risks drawing "ragged" boundaries too dependent on context-specific factors. Certain functionality may be lumped together as a domain for historical or incidental reasons, rather than reflecting any principled basis for domain definition. Yet we do need to consider stakeholder interests (not just in choosing the initial domain focus, but throughout the definition process) in order to avoid spinning into ungrounded abstract ideas (and usually arguments).

- *Domains vs. Systems.* A related challenge is a tendency, at least among software engineers, to drift into thinking about domains in "system" terms; i.e., a "domain" is equated in some

130

fuzzy and unspoken way to an object or component of a system. One consequence of this is that many contextual assumptions remain unexamined in our informal definitions of a given domain. In particular, for a domain corresponding to a sub-system level grouping of functionality, we may tend to consider domain functions only in the context of systems with which we are familiar. Later, when we try to use the domain model in other system contexts, hidden assumptions emerge to cause us trouble. On the other hand, a domain cannot be defined without some contextual assumptions. The definition process must somehow aid us in discovering and articulating those assumptions, thus making scoping decisions intentional and manageable.

Example. From the standpoint of a text editor system, an outlining capability is a "sub-system" of the entire application. It may be clear that there are important degrees of variability desired for the outliner subsystem of a certain product line of text editor applications.

But the Outliner domain has not been defined until we have at least considered the possibility of outliner functionality occurring in many other application contexts (presentation managers, file browsers, etc.) We are not required to expand our scope inappropriately; but we need to make explicit the criteria which determine the sorts of outliners in which we are interested.

Intermezzo: The Further History of the Example

So far in the process model we have presented an example scenario based on the hypothetical Persona Inc. software company. In the subsequent example text we will for the most part leave behind the Persona scenario and concentrate on aspects of the Outliner domain as reflected in commercial off-the-shelf applications like text editors, presentation managers, directory browsers, etc. The sudden curtailing of stakeholder issues in example discussions in *Define Domain* and beyond is not really by choice; it is more an artifact of the authors' constraints in preparing example material. On the other hand, a shift into definitional and technical modeling issues is not uncharacteristic of the *Define Domain* sub-phase and to this extent not altogether misleading. In addition, the scenario has motivated conditions in which the kind of data acquired for examples would be plausible.

Example. As a result of Persona's *Plan Domain* process the domain of "Outliner-based interfaces for PC-class applications" was selected. Because of the company's strategic interest in producing applications that can rival PC-class familiar interfaces, they decide on a project plan that involves three phases: first, a trial demonstration of domain engineering involving competitive analysis of outliner-style interfaces in commercially available applications; next, a pass that involves study of emerging outliner-based interface capabilities on the Web; and finally, a detailed requirements analysis of the interface requirements for a diverse set of specific anticipated contracts for the firm. The remainder of the examples reflect primarily the results of the first-phase project effort.

Approach

The approach to domain definition in ODM directly addresses the challenges outlined above. The domain is defined by establishing formal boundary criteria, cross-checking these with concrete examples, and clarifying the relationship of the selected domain to other domains of interest. By methodically testing boundaries with these complementary techniques, modelers are led to a set of definition *questions* that must be resolved to articulate the domain.

The *Define Domain* sub-phase thus constitutes a first "trial run" of domain modeling, a high-level excursion that can highlight or preview some major "fault-lines" that may reappear in processes performed in more depth during the more extended, more resource-intensive *Model Domain*

phase. The sub-phase can almost be thought of as performing a rapid prototype of the entire *Model Domain* phase to follow. The following paragraphs introduce the core concepts behind ODM domain definition techniques in more detail.

What are the qualities of a good domain boundary? It should be:

- *Communicable*: the boundary principles should communicate to people who did not participate directly in the definition process. Eventually the domain definition will serve as the "gatekeeper" for a domain model and/or asset base accessed by people from a wide variety of contexts. The domain definition may have a greater impact on the perceived quality of the model/asset base than the extent to which these resources are comprehensive or encyclopedic. A clear boundary will help potential users quickly and accurately determine whether or not what they are looking for is in the scope of the model/asset base.

- *Sharable*: this principle is closely related to the last one, but applies even to the team that collectively develops and works with the domain definition. A definition constantly subject to variant interpretations and confusions will have a detrimental impact on subsequent steps of the process. A stable and sharable boundary will serve as a working agreement that allows progress to be made. In fact, the challenge of coming up with this shared definition itself acts as a strong formative process for the modeling team.

- *Fine-grained*: Domain boundary issues turn out to be highly scalable. Thus definition criteria in turn need to be robust and flexible enough to apply at different levels of structure. It's a natural part of the process to only gradually tease out the lower-level implications of certain boundary decisions. This reinforces the notion that domain knowledge is "fractal" in nature, and that reasonable criteria at one level of description resolve into more complicated relations at more detailed levels. Definitions formed by accumulating a host of "exceptions" and special cases will tend to brittleness; more useful are principles of inclusion and exclusion that apply at different levels of analysis.

- *Evolvable*. The point of the process is not to arrive at some idealized "formally pure and correct" domain definition. This is clearly impossible, both theoretically and practically, and is not really the objective in any case. Rather, the definition should serve as an ongoing baseline that can be referred to and modified at various points forward in the process.

- *Surprising*. Though this may be a surprising criterion to find listed as a desirable quality for a domain definition, the rationale is pretty simple. The definition is intended to "wake us up" to tacit assumptions we make that we do not communicate or of which we are not aware. One way to know that we are paying attention to a definition is when we find that it does not match our informal intuition. This can happen in two complementary ways:

  — something we expected to be in the domain is not in the domain; or

  — something we expected to be out is in.

  In either case, a good definition triggers a process of reflection that results in new insight about the material, e.g., "Oh, I guess that does fit in the domain after all." Wee see connections and analogies or make distinctions and separations we were not able to make before wrestling with the definition. If the result is annoyance and a round of repair and adjustment of the definition, that is fine too; the definition *process* has served its purpose. A poor definition is a lazy definition, or one that lets us remain lazy.

Domain as A Set of Boundary Agreements

To arrive at a domain definition with the above qualities, the domain boundary in ODM is articulated through a series of inter-related boundary decisions. Most have strategic impact, but the decisions are also inter-dependent and impose their own constraints. Central to the ODM approach is the idea that a domain definition is a *shared set of boundary agreements*. We tend to visualize a "boundary" as a continuous line, a "skin" around a coherent object with empty space between the object and other objects. In the ODM perspective, by contrast, a domain boundary is formed by multiple, discrete boundary *decisions*, each involving negotiation and reflection around a particular issue or question. Exhibit 43 contrasts these two intuitive pictures.
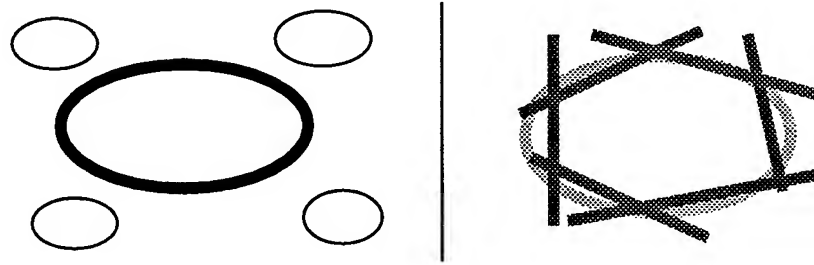
Domain as a bounded object ... vs. ... a set of boundary decisions

Exhibit 43. Different Intuitions of Domain Boundaries

While the particular boundary decisions to be determined will vary with each domain, the *Define Domain* sub-phase is structured to reflect certain recurring *kinds* of boundary issues that must be considered. By distinguishing these different kinds of issues, the process helps ensure that each type of issue has been dealt with explicitly in the definition process, and helps avoid certain sources of confusion. The process also sequences the order in which the decisions are considered, making the steps more tractable.

Results

The primary output of this sub-phase is the DOMAIN DEFINITION, which gets continually refined and updated throughout the domain engineering life cycle. The DOMAIN DEFINITION defines the relative scope of applicability for the domain, the scope of features included within the domain, and its relation to other domains.

Process

As shown in Exhibit 44, the *Define Domain* sub-phase consists of three main tasks:

- In *Bound Domain*, we decide which exemplar systems as a whole exhibit the type of functionality deemed to be "in the domain." Here, the domain is viewed as a "class" of systems exemplified by a specific, enumerated "set" of systems. The domain boundary is formalized, both in terms of criteria for inclusion and exclusion or *defining rules*, and in terms of identified member systems or *exemplar systems* which provide an empirical cross-check to the formal bounding criteria.

- In *Focus Domain*, the domain boundary is further refined by determining what aspects of the given exemplar systems fall within the scope of the domain. In the first task, we looked for

Exhibit 44. Define Domain IDEF₀ Diagram
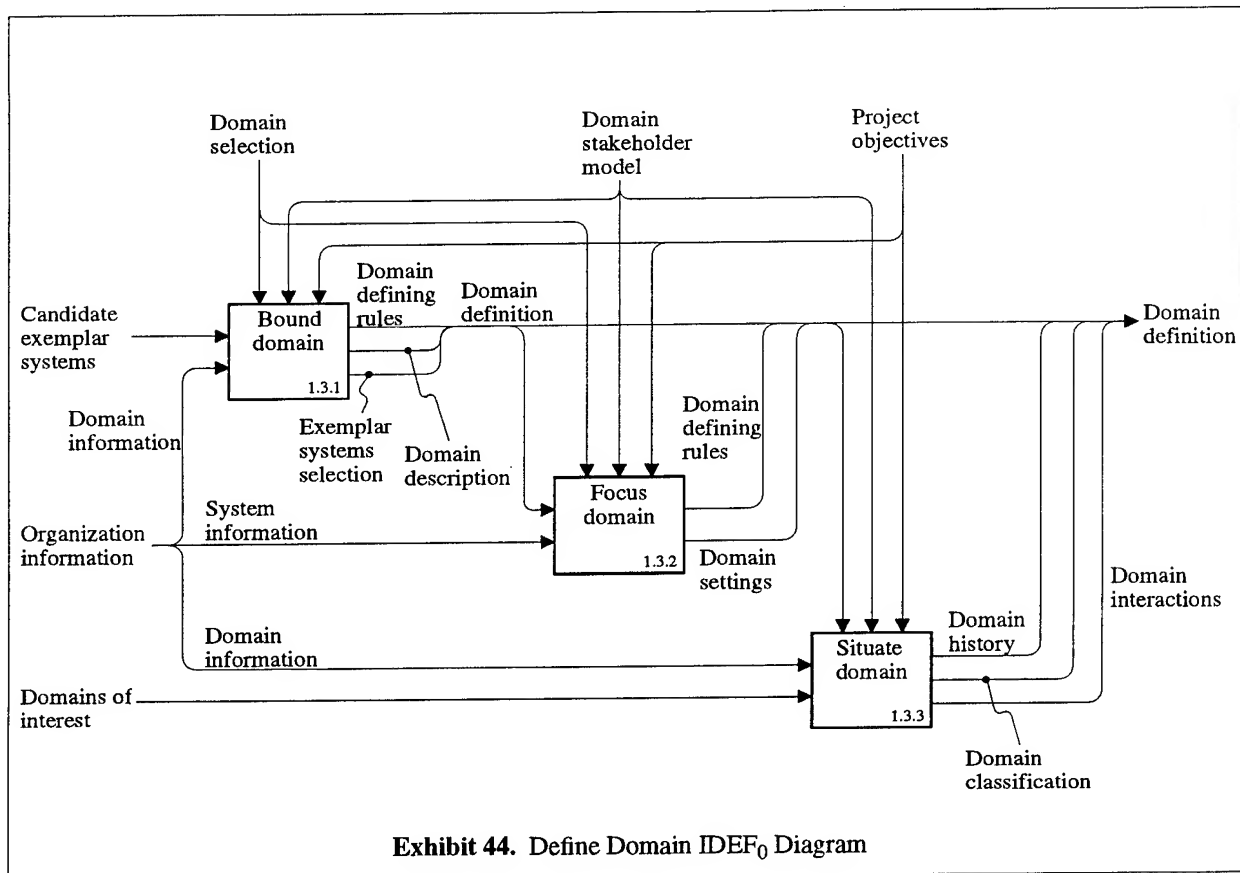
the "systems in the domain." Next, we're looking for the "domain in the system." That is, we decide what *portion* of the systems in the domain belong to the domain scope. This means identifying key interfaces from domain functionality to related system capabilities that are to be considered external to the domain.

Though a bit simplistic, it is helpful here to think of a system as characterized by bunch of attributes, qualities, principles, or characteristics (called, for convenience and informally at this stage of the process, "features"). The definition selects a set of these features as "relevant" to the domain. Those features thrown into the domain definition's "feature bin" constitute the intersection, as it were, of the domain's "flashlight circle" in the overall system functionality. Some domains will throw a distinct circle of illumination. Others (like a scene under a black light) will illuminate disparate system aspects that fall within the common criteria for domain inclusion.

- In *Situate Domain*, the results of the first two tasks are integrated in order to identify qualitative relations between the domain of focus and conceptually related domains. Having determined these complementary boundary decisions, we can now name what lies outside the domain boundary, both with respect to systems excluded from the domain's set of exemplars, and features (e.g., system functions) excluded from the domain scope. We orient the domain in terms of its historical context, broader and narrower domains (providing subsets and supersets of domain functionality), related (or "neighboring") domains, and various application contexts in which domain functionality appears.

This primary sequence for the tasks in *Define Domain* reflects a progressive degree of information detail needed about exemplar systems in the domain. *Bound Domain* involves boundary decisions for exemplar systems as a whole. *Focus Domain* requires greater examination of the structure of

particular exemplar systems and an understanding of how the domain scope "intersects" this structure. Finally, *Situate Domain* may require some investigation of the historical background of the domain's evolution and related domains.

Articulating the natural boundaries of the domain of focus is a prerequisite to an informed decision about whether to:

- continue on to the full *Model Domain* phase; or

- iterate back to the *Scope Domain* sub-phase to

  — *refine* or *correct* the selection criteria and derive a more focused domain; or to

  — perform a new selection process for a different domain of focus. This can be useful for conducting a high-level "scan" of promising domains within a given ORGANIZATION CONTEXT, prior to going on to modeling for any of the domains.

The primary distinction between *defining* and *modeling* activities is that we concentrate as much as possible on criteria for what is in vs. out of the domain, deferring consideration of how phenomena within the domain are differentiated. In practice, these activities will tend to interweave, but there is benefit in preserving the distinction. One discipline needed to successfully perform *Define Domain* activities is to notice when you have started differentiating within the domain, and deferring this until the definition has stabilized.

Guidelines

Sequencing. The tasks within *Define Domain* need not be carried out in strict sequence. The documented ordering provides some risk reduction, in that earlier boundary decisions help limit the amount of information examined in subsequent activities. But there will be a natural tendency to interleave the activities of making boundary decisions and qualifying or naming the related or neighboring domains that lies beyond that boundary. This is not a problem, as long as we don't try to get closure on the nature of domain interrelationship at this point. Each successive boundary decision changes the overall picture.

> *Example.* In the example above, we will probably have seen a system that exhibits the multiple-parent heading view capability. Suppose we have decided to exclude systems with this feature from the domain; our system thus becomes a **counter-exemplar**. At this point, it is natural to ask "What kind of application is this, if it is not an Outliner application?" As we try to give the "other side of the boundary" a provisional name, we step into the territory of *Situate Domain*.

Breadth vs. depth. Two broad "process styles" could be described for this sub-phase:

- breadth-first: following the structure suggested by the tasks and identifying most boundary issues at each stage before moving on; or

- depth-first: following the "thread" of an individual boundary issue through the various tasks and repeating this process for other issues.

Since the definition process can be viewed as a series of discrete boundary decisions, it can be useful to document the process of these boundary decisions on an ongoing basis. As decisions about domain boundaries shift in the course of the project, such a report can capture process history, decisions and rationale for the shifts. This information is also valuable for future lessons learned, and for validation and modification of related workproducts to assure consistency of all

ODM workproducts during modeling changes. It can also help to flag process breakdowns, like a tendency to revisit the same boundary decisions over and over again. This is particularly valuable when there is changeover in project staff over the lifetime of the effort.

Innovative domain boundaries. As noted in the *Select Domain* sub-phase description, domains can be selected emphasizing either status-quo or innovative values. Innovative domains force us to group functions and capabilities in novel ways. The more innovative the domain selected, the more critical it is to allocate sufficient resources to the domain definition effort. But familiar domains can be deceptive as well! Don't assume simple boundary issues because you are dealing with a named domain that everybody already uses.

Boundary vs. focus. The most difficult concept to keep clear in this task is the distinction over clarifying the domain boundary (determining whether systems with a given capability are specifically *excluded* from the domain) vs. the domain focus (determining whether we are excluding the capability from the set of features we will consider part of the domain scope).

> *Example.* In the Outliner domain, there are some systems that provide support for headings that have multiple parents (i.e., beyond strict hierarchical structures). Suppose we decide to exclude applications that support headings with more than one parent from the domain. Thus a system that supports multi-parent headings would be considered a ***counter-exemplar*** for the domain. On the other hand, we may decide that such systems still present Outliner functionality in the sense of our domain definition. However, though such a system would then be considered an exemplar, that specific capability would not be within the scope of the domain.

5.3.1 Bound Domain

The *Bound Domain* task is the first task of the *Define Domain* sub-phase. Since the *Define Domain* sub-phase serves almost as a "domain modeling prototype" process, this task reflects the transition from the primarily strategic front end into the formal modeling activities in the middle of the life cycle.
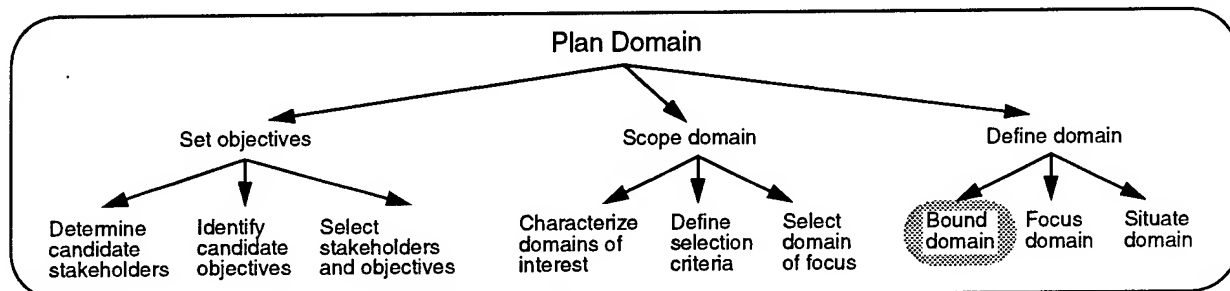
Exhibit 45. Bound Domain Process Tree

The primary goal of the *Bound Domain* task is to refine the informal notion of the domain included in the DOMAIN SELECTION into a set of explicit criteria defining what systems are in or out of the domain. We define what is entailed by the selected domain, and clarify what is in and out of the domain, by specifying a verifiable, bounded set of principles and examples.

A key tradeoffs in this task is balancing breadth and focus. The temptation here is to be too ambitious in terms of the generality of the domain to be defined. At this stage we are focusing on shared criteria of domain exemplars (common elements) and are not yet confronting variability within the domain in any systematic way. We may therefore tend to cast the net too wide in terms

of the domain's intended scope, only to find later that the definition encompasses a range of variability that cannot be effectively exploited for reuse. The challenge is how to anticipate the "right" extent up front.

Approach

The techniques employed in this task are intended to help elicit some of the hidden contextual assumptions that creep in when groups adopt "working" definitions. The techniques may lead to arguments, but (provided everyone stays friendly) this is a sign of the process doing its job, not a breakdown. Disagreements, if pushed through, will often trace back to either a stakeholder issue that needs to be resolved, or a terminology issue that can be clarified without anyone "giving ground." The basic principle could be stated as follows:

Undocumented assumptions cause problems. Documented assumptions create focus.

The domain is defined in a variety of complementary ways that serve as cross-validation. These are described briefly below.

Domains as Sets Vs. Classes

Domains can be thought of as either *sets* or *classes* of systems. Defining a domain as a set of systems means that domain membership can be completely described with reference to some specific collections of systems. For example, an organization maintaining a group of twenty legacy systems in a common application area could treat those twenty specific systems as a domain. It would not matter whether there was a consistent set of characteristics to the systems; the domain would be defined extensionally. This can make sense in situations where domain engineering is closely tied to specific system goals: for example, to consolidate three similar systems inherited as part of a merger or acquisition and reduce the overall footprint of the consolidated system would mean studying the commonality and variability across those three systems only. Often the explicit criteria for a "domain as set" are organizational boundaries or historical connections: i.e., a typical way of defining a domain might be "all systems maintained by group X" or "all systems customized from the original system Y for different customers in the last ten years."

Defining a domain as a *class* of systems means having some set of rules or criteria for defining what systems are or are not members of the domain. The class of systems approach means the domain extends to a potentially unlimited set of systems. New systems can come along and be judged as belonging or not belonging to the domain.

The essence of the ODM approach to domains is the realization that these two ways of defining systems are always closely intertwined. Usually when we group a set of systems from the "set" point of view, we have some principles or rules in mind of what those systems have in common (besides their common organizational linkage; in fact, typically the common aspects are why the systems are developed or maintained out of the same division). Conversely, whenever we state a seemingly general set of criteria we have some set of example systems in mind.

Extension and Intension

Extensional definition involves designating a set of systems as examples of the domain (called *exemplar systems*). These are documented in the EXEMPLAR SYSTEMS SELECTION. The extensional definition provides concrete examples that everyone can relate to as a common basis for understanding, hence offers empirical validation that there is shared understanding of the domain.

Note: Classifying a system as an exemplar does not constitute a plan to study it in detail. The set

137

of exemplar systems to be studied intensively, the REPRESENTATIVE SYSTEMS SELECTION, will be determined in the *Plan Data Acquisition* task within the *Model Domain* phase.

Intensional definition involves rules of inclusion and exclusion of systems with respect to the domain. *Intensional* is a term freely adapted from the field of logic to denote boundary criteria that take the form of features, attributes or rules against which an arbitrary candidates can be tested to determine domain membership. Intensional rules are used to classify new candidate exemplars as being inside or outside the domain.

Essential vs. Accidental

The power of the ODM approach lies in the ability to cross-correlate the extensional and intensional ways of defining the domain. Since our categories are usually formed with reference to examples, the challenge in coming up with a good intensional rule is to ask what property a given example has that makes it an exemplar of the domain. The rules should, as much as possible, concern *essential* and not *accidental* features of the domain; i.e., is the rule merely an observation of a shared attribute of the examples considered so far, or is it a required attribute that would apply to new cases considered as well?

> *Example.* In the Outliner Domain: Suppose that all outliner systems examined used a triangle icon as an indicator for a heading or topic within the outline. Would this be considered an "essential" feature of our domain? If we found an outliner using a different icon, would we exclude it? Probably not; therefore this would be deemed an *accidental* feature.
>
> On the other hand, consider the feature of expanding and collapsing the view of sub-headings beneath a given heading. This would probably be considered an *essential* feature of the domain; i.e., it is a feature without which a given application would not exhibit "outliner-ness" (as we are choosing to define it).

A stable definition requires specifying both what is *inside* and what is *outside* the boundary of interest. This is reflected in defining explicit rules of both *inclusion* and *exclusion*.

> *Example.* Outliner systems that handle multiple parents have a very different set of problems to address. For this domain analysis effort, we want to specifically *exclude* systems that have this feature. We are interested in studying that set of applications trying to solve a simpler data structuring problem.

Workproducts

■ DOMAIN DESCRIPTION

A refinement of the informal domain description included in the DOMAIN SELECTION, based on insights gained in this task. This serves as a consensus document for the project team about what the domain includes. As such, it is an implicit input to almost every activity downstream in the domain engineering life cycle. It can also be used to orient new project members, in presentation to management and to external audiences, and to communicate the intended scope of the domain of focus to informants during the *Model Domain* phase.

The DOMAIN DESCRIPTION can be a simple textual document, ranging from a few paragraphs to a page or two. It should include the following:

* *Domain name*: The name may be refined with each domain boundary decision, although it will not be possible to adequately convey all criteria in the name. (Sometimes an acronym is

a good compromise to a long, cumbersome phrase, particularly if some humor can be employed.)

- *Audience*: The description can be targeted to experts and/or novices in the domain. It can include exemplars that will be familiar to readers, and textual statements of defining rules. However, it is primarily an aid to comprehension; it is not meant to convey the formal boundary rules for the domain.

- *Informal definition*: This can include some illustrative examples, the distinctive features of the domain, the purpose of domain systems, typical usage settings, etc.

■ DOMAIN DEFINING RULES

Rules of inclusion and exclusion for domain membership and the logical relationship between these rules. A *defining rule* is a statement about various combinations of characteristics or attributes that imply membership in the domain.

The rules can be written informally or formally; for example, they could be complex predicates built out of atomic statements composed with logical operators (and, or, etc.) or simple natural language statements. They should distinguish the following categories of rules, however:

- Inclusion vs. exclusion;

- Essential vs. accidental;

- Linkage to exemplars (as described below).

A worksheet template that can be used to cross-validate DOMAIN DEFINING RULES against exemplar systems is shown in Exhibit C-8, EXEMPLARS/DEFINING RULES MATRIX.

■ EXEMPLAR SYSTEMS SELECTION

A list of exemplar systems for the domain that complements the DOMAIN DEFINING RULES by identifying systems judged to be members of the domain. The EXEMPLAR SYSTEMS SELECTION should document *all* systems that planners have considered with respect to domain membership. A worksheet template that can be used to cross-validate DOMAIN DEFINING RULES against exemplar systems is shown in Exhibit C-8, EXEMPLARS/DEFINING RULES MATRIX.

We start with a set of *candidate exemplar systems*—the set of systems that have been considered as "possibly" belonging to the domain. As a result of the gradual refinement of the boundary criteria, each candidate system considered is allocated to one of the following categories, reflecting degrees of membership in the domain:

- *Exemplars*. Systems with sufficient features for membership in the domain according to the defining rules. Exemplars have several important sub-categories:

  — *Core exemplars* are systems judged to strongly define or characterize the essence of the domain (e.g., an established industry-lead product). Identified characteristics of core exemplars will typically be strongly correlated with important DOMAIN DEFINING RULES.

  — *Key exemplars* are systems required to be covered by the domain definition for strategic reasons, typically due to the interests of some key stakeholder, i.e., "If we're going to focus on this domain then we have to consider X as an exemplar." They may not have strong correspondences with principles for inclusions or exclusion beyond their individ-

ual case. The term "key" here refers to the strategic or stakeholder-driven importance of these exemplars in the definition process. Key exemplars are distinctive because if defining rules wind up excluding them, the rules must change!

— There can (and should) be other exemplar systems in the set as well. The set need not be exhaustive.

• *Borderline exemplars.* Systems that have ambiguous criteria, i.e., that provide evidence for both domain membership and non-membership. but not sufficient evident for either. Borderline exemplars may be focal points for disagreements within the domain planning team. Such conflicts are often symptomatic of a domain boundary issue that may be iteratively explored and resolved via adjustment of the defining rules and/or exemplar set.

• *Counter-exemplars.* Systems with features identified that provide sufficient evidence of non-membership in the domain. Since listing all possible counter-exemplars would be in theory an infinite (and unenlightening) task, the point is to look for candidate systems that lie "just outside" the border of the domain.

The most interesting counter-exemplars are those that match many and fail only a few defining rules for the domain. Chosen in this way, each counter-exemplar helps to validate the necessity and/or sufficiency of various defining rules. Documentation of counter-exemplars also helps subsequent users of domain modeling results distinguish systems never considered for inclusion by the project team from those considered and rejected for specific reasons.

When to Start

Domain bounding activities can start as soon as:

• The project team has selected the domain. An informal description of the domain of focus has been completed. The bounding process works by refining the shared informal notion of the domain.

• Stakeholders have committed to the selected domain. If the bounding process is attempted for an informally defined domain without real stakeholder commitment behind it, it will not be the same process. (This may be useful as a learning exercise.)

Inputs

• CANDIDATE EXEMPLAR SYSTEMS. Used here to develop the exemplar system selection.

Note: Candidate exemplars are the "projection" of the systems of interest judged to have some overlap with the domain of focus. This scoping down is significant. We want counter-exemplars to be systems excluded because of some "relevant" defining rules for the domain of focus. That is, bounding the domain will elucidate "things we might have thought were in the domain but have decided will not be." If we worked from the original systems of interest then counter-exemplars could include systems having nothing "interesting" to do with the domain of focus at all.

• DOMAIN INFORMATION. Information about the domain known by the domain informants. Used to identify domain terminology relevant to definition of the domain. DOMAIN INFORMATION is gathered informally at this stage.

Controls

• PROJECT OBJECTIVES. These are needed as a reference when boundary issues arise. Refer-

ence back to overall project objectives helps keep the bounding task focused on the definition of the domain that is most useful in achieving PROJECT OBJECTIVES.

- DOMAIN SELECTION. The informal domain description is used as a basis for the defining rules of inclusion and exclusion for the domain. Documented linkage to specific systems that formed part of the rationale for the domain selection also becomes input to determining the EXEMPLAR SYSTEMS SELECTION.

- DOMAIN STAKEHOLDER MODEL. Assists in the identification of exemplar systems based on the links those systems have with stakeholder organizations. Some domain boundaries may be defined in organizational terms (e.g., "the set of all inventory control systems maintained by Division X of Company Y.")

Activities

The activities in this task can be structured in terms of the following polarities:

- Working intensional against extensional definition (rules vs. exemplars);

- Working strategic or stakeholder-oriented factors against intuitive or conceptual factors (key vs. core); or

- Working informal, intuitive against formal descriptions.

The activities do not reflect a strict sequence; a variety of approaches could be used.These activities can be performed largely in parallel. For example, two independent groups could develop the DOMAIN DEFINING RULES and EXEMPLAR SYSTEMS SELECTION, iteratively checking the resulting workproducts for compatibility.

➤ Identify key exemplars

Typically, when we come up with an informal definition, we have certain examples in mind. The objective of this activity is, first, to get these "tacit" examples documented. These correspond to the "intuitive" exemplars for the domain.

Next, consider the stakeholder interests factored into the PROJECT OBJECTIVES and the DOMAIN SELECTION. Assess whether this places certain constraints on what exemplars "must" be part of the domain. These are what we refer to as *key* exemplars, in the sense that "this system must be an exemplar in order to ensure alignment with stakeholders and objectives." Document the key exemplars in the EXEMPLAR SYSTEMS SELECTION.

As a validation cross-check, look for the key exemplars in the CANDIDATE EXEMPLAR SYSTEMS. If there are key exemplars that do not appear in this list, you may want to ask why: Were they missed from the original systems of interest? Or were they improperly characterized in terms of their overlap with the domain of focus, hence not included in the CANDIDATE EXEMPLAR SYS-TEMS? This may help to point out some issues that need resolution)

➤ Infer rules from key exemplars

If appropriate, you may infer a defining rule from key exemplars. Typically such rules will concern organizational scope rather than functional properties or attributes of the systems (e.g., "Exemplar systems in this domain must be systems being maintained by Division X.") If the resulting rule is so specific that it can't be used to infer anything about the exemplar status of any other systems (e.g., a rule like "System X must be an exemplar...") don't bother with the rule.

➤ <u>Identify other candidate exemplars</u>

Consider other systems that are likely candidates for membership in the domain. As an aid to seeking out interesting border cases for validation and completeness, you can use the following checklist to brainstorm other candidate exemplars.

The following informal typology may be useful as a starting point for brainstorming candidate exemplar systems for the domain. This typology is intended to elicit domain boundary conditions in both historical and functional terms.

Historical Exemplar Roles

Domains have their own "life cycle" which may be realized in a family of applications developed within a single organization, or a "genre" of commercial applications that emerge as products, compete in a marketplace where certain features and technologies form competitive differentiators. To discern the historical relations for the domain of focus, consider the following specific roles as a "starter list":[3]

- *Pioneer.* What system is generally accepted as the earliest occurrence of the domain functionality? This is often where a basic technology was introduced or first applied in a new application area. Often, these systems may not have been in widespread use; they may have been the discoveries of a small group of enthusiasts.

  Example. In the Outliner domain, ThinkTank was an early idea processor generally credited in the literature with introducing the concept of interactive outline editing. Interestingly enough (though perhaps not atypical) the outliner functionality was not the central focus of this product. The developers of this product assumed that their "value added" would have to be a series of specific *content* presented in extensible outline form. Outlining functionality was later identified as a capability with its own inherent usefulness, which led to later developments in the domain.

  (Preview.) Note that any information acquired to establish these data points is only provisional at this point. The roles are primarily intended to tease out implicit boundary conditions. Later in the modeling effort, we will discover that this historical picture is an oversimplification. The initial iteration of the Outliner domain modeling effort focused mostly on publicly available product information.In fact, early versions of outliner functionality were developed from specialized programming language editing environments (e.g., for Lisp) on high-end engineering workstations. This data may only have been obtainable through direct contact with developers.

- *Model-T.* What system is accepted as the basic version of domain functionality? This implies a few historical developments: spreading to a larger market, increased and better-defined functionality, emergence of a recognized distinct "niche" for the domain.

  Example. In the Outliner domain, Acta Advantage was an early product that provided standalone outlining capabilities without extensive word-processing functions.

- *Classic.* What is the standard example of a system implementing the domain functionality?

  Example. In the Outliner domain, the outline mode embedded in Microsoft Word's text editor

---

[3.] Note that this "starter list" is itself an example of the use of an analogy domain as a modeling technique. In this case the analogy domain is that of commercial automobiles. This analogy might not be culturally appropriate or evocative for all readers, e.g., readers outside the U.S. or devotees of public transportation.

might be considered a standard point of reference for outlining capabilities.

➤ Identify descriptive statements

The goal now is to derive statements that are descriptive of systems in the domain.Not all these statements will turn out to be necessary for membership; the goal is to tease out distinctions between essential and accidental properties of the domain. There are two major sources for these statements:

- Highlight and list individual statements in the INFORMAL DOMAIN DESCRIPTION that you believe will hold true for all or most exemplars of the domain. Although it may be doubtful that this statement, in its loose form, will turn out to be a precise defining rule, we identify it as a separate statement with which to work. As with any textual analysis or requirements analysis, statements can be decomposed or simplified as needed. Statements about the purpose of the domain will suggest at least some high-level functionality provided to some stakeholders.

  Example. In the Outliner domain, a general purpose statement might be, "Outline processors allow writers to manipulate text in units structured according to hierarchical outlines."

- Using the CANDIDATE EXEMPLAR SYSTEMS list, generate a set of statements that would appear to be true, or false, of all systems in the domain. Pick some convenient preliminary name that means "system in the domain" to use in the statements. This is only a placeholder; we will refine this name at the end of the task.

  Example. For the outliner domain, we might generate these statements:

  Outliners give you a way to view lots of data in summary form.

  Outliners let you hide levels of detail.

  Outliners help you organize your thoughts.

➤ Consolidate descriptive statements

Integrate and consolidate the descriptive statements derived in the previous activity. If statements appear to be reworded duplicates, try to consolidate them or split their scope of concerns more clearly; but don't worry too much at this point about overlaps in the statements. Statements that appear to conflict are a more serious problem; they may represent an unresolved stakeholder issue, or an unclear objective.

This closes the informal part of this task. The idea is to catch obvious problems that can be seen informally before moving on to formalize the definition.

  This is another general principle followed in ODM; as much as possible, use informal techniques to sift out "noise" from the material that will be worked with formally.

➤ Classify descriptive statements

For each descriptive statement identified, evaluate whether it appears to describe an essential or an accidental characteristic of systems in the domain. Note that while this is easily said, this activity involves hard conceptual work and is the most significant design decision made in the *Bound Domain* task.

Assess whether the statement is true for all the systems currently classified exemplars.

- If not, then the statement is of a form that eventually (in the *Model Domain* phase) might become a ***differentiating feature***. In this case, we want to work our way up to a more general feature that would be true of all systems in the domain. To get to this more general feature, find a counter statement true of a different exemplar; then try to find a more general statement that subsumes both the original and the counter-statement. Once you have this statement, you can check again whether it applies to all currently

- If the statement is true for all systems currently deemed exemplars, try to find a system that matches all defining rules selected so far, yet does not satisfy the statement in question. Now the critical question is: *is this system an exemplar or not?* This will involve checking the strategic or stakeholder basis for the answer (e.g, will I have a domain boundary that will help meet the project objectives?) against the intuitive or conceptual basis for the answer (e.g., the boundary will be "cleaner" if I include/exclude this system...)

If every system you think of would not be an exemplar, you have determined the statement to be an essential feature of systems in the domain, hence the basis for a defining rule. If an exemplar can be discovered (or at least hypothesized) that is in the domain but for which the statement does not hold, then the statement is deemed an accidental feature. The exemplar should be added to the CANDIDATE EXEMPLAR SYSTEMS, and the descriptive statement added to the differentiating statements.Leave the differentiating statements alone. They will be formalized later, unless some of them need to be skimmed off to augment or repair the formal definition as it grows.

Descriptive statements deemed accidental can be handled in one of two ways:

- They can be determined to be consequences of one or more defining rules. That is, by themselves they are not definitional; but given certain other defining rules the descriptive statements will always hold. These statements are worth capturing, because they represent inferences that can be made about the domain. However, they are best kept out of the defining rules list, so that this list represents the minimal "hoops to jump through" in testing exemplars.

- They can be determined to be *necessary* but not particularly *relevant*. This usually means they are a consequence of some criteria for bounding the domain that do not involve interesting features with respect to the domain. E.g., suppose a boundary criterion involves "All systems in domain X run on a Unix platform." Then any number of facts about Unix-based systems will be true about systems in the domain, but these facts may not reveal anything else that sheds insight on the nature of the domain itself.

➤ Determine and document defining rules

Based on the decisions made in the last activity, select the committed set of defining rules and document them in the DOMAIN DEFINING RULES workproduct.

Example. For the Outliner domain, defining rules selected include the following (excerpt only):

— Outliners that support multiple parents are excluded from the domain;

— Outliners must support more than a two-level nesting of heading levels (this rules out certain candidates like to-do list managers with click to hide and reveal a single level of additional data);

— Outliners that deal with text-based content are included in the domain.

➤ Cross-validate rules against exemplars

A worksheet template that can be used for this activity is shown in Exhibit C-8, EXEMPLARS/ DEFINING RULES MATRIX. The result of this activity is a classification of the candidate exemplars as core, counter-exemplars, or borderline exemplars. Every system in the EXEMPLAR SYSTEMS SELECTION should have been characterized in terms of defining rules. Rules should match those that allow (if not require) inclusion in the domain. Adjustment of the defining rules or of exemplar classifications may be required.

➤ Refine domain description

As each boundary decision gets made, modify the textual DOMAIN DESCRIPTION appropriately. As a further test that the overall principle of the domain is staying focused, put some care into naming the domain with a suitably descriptive phrase. Each term in the name can be validated by specifying what alternatives it excludes as well as includes (linking to the rules and exemplars that have been explored), and thus why the qualifier or conditional is essential to the definition.

> *Example.* For the Outliner domain, although "Outliner" is used throughout as a short-hand reference, a possible name for the domain might be: "Interactive Hierarchical Text-Based Outlining Browser/Editor Functions." Let us examine the name in more detail:
>
> *Interactive:* There are a number of batch-oriented capabilities that provide options for *formatting* documents in hierarchical outline style; e.g., certain sets of typographical markup commands. These might be an interesting related (and more confined) domain to study. But the focus of this domain is on applications that provide dynamic interactive viewing, modifying and presentation capabilities.
>
> *Hierarchical:* There are a number of interactive applications that provide capabilities critical to "outliner-ness" such as summary views or hiding and collapsing sub-structure. However, the ability to perform these operations on hierarchically oriented content is a bounding criterion for the domain of focus. This excludes a number of programs like flat list managers (e.g., some to-do list managers, a flat Web-based bookmark list manager, etc.)
>
> *Text-Based:* There are a number of applications that provide ways of viewing hierarchical structures of various kinds, such as graphic tree displays, etc. Essential to the notion of outliners we wish to explore is the idea that the structural items being managed are based around text. Note that this still encompasses applications as variable as a file system manager (the outline capabilities operate on text strings denoting names of directories and files), outlining text editors, stand-alone outliners and "idea processors," specialized outliners like structured to-do list managers, and even slide presentation managers. It will turn out that this weighting towards text-oriented displays has pervasive impact on the range of features discovered.
>
> *Outlining Browser/Editor:* In true German-language fashion, we get to the crux of the name only towards the end! Outlining here is the name selected to denote the basic viewing, collapsing/expanding, and modifying operations that characterize this class of applications. There is a bit of ambiguity here, in that we will consider some variants that do not allow editing, but only browsing.

When to Stop

You can stop *Bound Domain* activities when:

* Key exemplars are all accounted for in the set.

- The definition process seems to have stabilized.

- The resulting exemplar set and defining rules are consistent. For example, all members of the EXEMPLAR SYSTEMS SELECTION satisfy the domain defining rules. There is at least one system identified in the EXEMPLAR SYSTEMS SELECTION for every domain defining rule.

- The definition seems to capture the intuitive notion of the domain. The defining rules appear necessary and sufficient to characterize whether or not given systems are within the domain.

Guidelines

- <u>Don't try to get it perfect</u> at this point; it won't be. Subsequent steps may result in iterating or refining the boundary criteria. The goal is to establish a starting point that can be evolved.

- <u>Use borderline categories to limit thrashing</u>. If the process won't terminate because of an unresolved issue, capture it via the borderline exemplars, note the conflict and move on. After all, open boundary issues are themselves part of the domain as "shared and negotiated set of boundary agreements." Don't try to work stubborn issues once identified; freeze any particular boundary discussion if you have flip-flopped more than once.

- <u>Do not try to systematically elicit differentiating features</u> for exemplar systems at this stage. We want to know what is in and out of the domain, not all the different kinds of things that are in. That will be the goal of the *Model Domain* phase.

  Some observation of differentiating features will occur naturally. Capture them for use later but keep them separated from the definitional features and don't get distracted by them. The only reason to enumerate differentiating features here is if that's the only way to convey the definitional intent.

  Example. Suppose that the systems of interest are implemented on a variety of software platforms. For strategic reasons, you want to consider only those systems on three of the platforms, A, B and C. In this case the only feasible way to define the domain boundary conditions may be with a rule that enumerates the included features: "The domain includes systems running on A, B or C platforms." This rule is really only a "composite" of three differentiating features, but there may not be a better way of describing the principle.

  On the other hand, sticking with this example, it might be helpful to probe a little bit: why is it that we are interested in platforms A, B and C only? Because they are the platforms of most immediate interest in the near term for the organization's targeted customer base? Then this latter statement is what really belongs as the defining rule. In this way, the strategic intent and stakeholder interests get captured explicitly in the definition.

- <u>Defining rules should be relatively independent.</u> Try to avoid rules that are special cases or inferable from each other. Each defining rule may eventually become the root of a feature model that covers the differentiation within the domain relative to that feature (which will by definition be common to all exemplars). Relatively independent domain defining rules will facilitate this later task.

- <u>You are making decisions about definitions</u>. If you lapse into arguing about "what the domain *is*" instead of "what we want to define the domain to *be*" the process will tend to break down.

  There is a helpful model to use to short-circuit such process breakdowns, illustrated in Exhibit 46. This is an example of an ODM "starter belief model" that maps typical beliefs or "stances" people take when having conversations and negotiations about boundary issues [LIBR96]. You can use this belief model to reflect on your own stance in the definition pro-

cess, to diagnose where other people are coming from, and to pick tactics for how to shift conversations to get them unstuck.

Naive Realist. The basis of the model is the idea that there is a basic un-reflective stance people instinctively take in such discussions that tends to freeze rigid positions, polarize issues, and keep interactions heavily weighted towards advocacy rather than dialogue. We will term this stance the "naive realist" stance and paraphrase the underlying belief as: "Domain X is clearly about xxx." The domain is "reified" as a kind of pre-existing object or category not open to debate. (And, of course, if there is a disagreement the naive realist is usually pretty sure that he or she is right!)
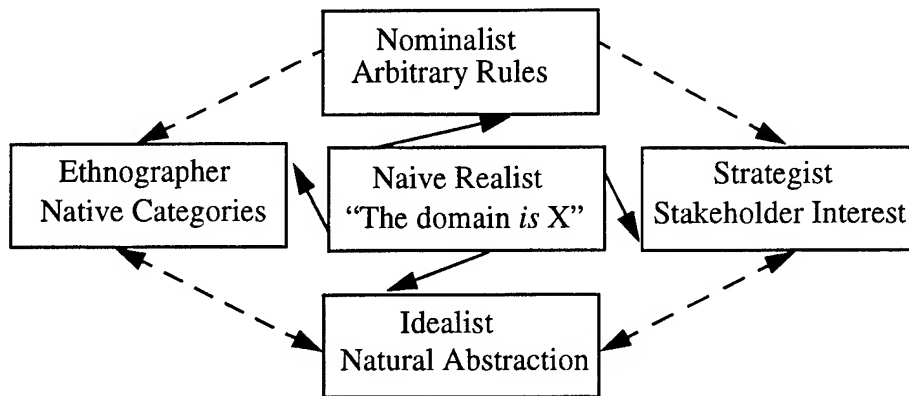
Exhibit 46. Typical Stances in Domain Definition

The model suggests at least four complementary ways in which the "naive realist" stance can be shifted. Different people will feel stronger affinity with certain of these other stances, but all are a step towards reflectiveness in comparison to the naive realist. Aspiring domain modelers would do well to practice intentionally "adopting" all four stances in succession with respect to any given boundary issue being examined. They are as follows:

— *Nominalist.* Domain definition is viewed as an arbitrary "game" with some formal rules and ways of checking results for consistency.

— *Strategist.* Domain definition is a reflection of "technical politics" and boundary decisions can be traced back to motivation grounded in some stakeholder interests. Strategists ask, "What *should* the domain be?" or "What do we want the domain to be?"

— *Idealist.* Domain definitions should reach for the "natural abstractions" that represent coherent, clean concepts with good separation of concerns and intuitive clarity. Coming from this stance, there may not be a "right" domain definition in the naive realist's terms but domain definitions (and later, models) can be judged on aesthetic grounds. These are the folks searching for the Platonic ideals in the domain.

— *Ethnographer.* These folks are willing to look at the domain definition as reflecting a "native category" used by practitioners within a given community. They will tend to ask the question: "What does the term X mean as a domain within this organization or this group of engineers?"

    Naive realists are on their way to adopting an ethnographic stance if they can get to the "In my village..." view of things.

Once someone has made the shift to one of these four stances all kinds of further shifts

between them may take place. Some notions of these are suggested by the dotted arrows in Exhibit 46. These are not as critical to use of this model as getting to the first transition.

5.3.2 Focus Domain

In the previous *Bound Domain* task, we have determined rules of inclusion and exclusion for domain membership and enumerated specific exemplar and counter-exemplar systems. In effect, we have viewed the domain as a set or class of systems. (The exemplars suggest the set, the defining rules the class.)
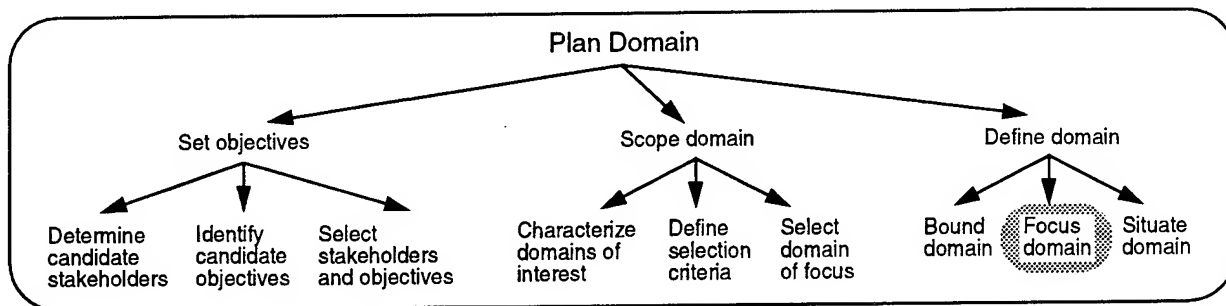
Exhibit 47. Focus Domain Process Tree

This bounding step would be sufficient if we were always dealing with the entire system scope as the intended scope of our domain. But so far, all that the status of "exemplar" means is that relevant domain functionality occurs somewhere within the system in question. The entire system isn't the exemplar, but it contains some "exemplary stuff." The domain of focus will be some specific set of features shared by the exemplars. To get at this defining set of features, an additional definition step is required.

It can be very expensive in resources and expertise to get at system details. Without a basis for determining what are and are not domain-specific features, any attempt to collect domain information from different sources (particularly by different investigators) is likely to wind up with highly disparate data. For example, in looking through a requirements document for those requirements specific to the domain of focus, not all such requirements will be clearly designated in the document structure or will be easily differentiated simply by use of certain key terms. Relevant requirements may be scattered throughout the document structure. In general, this same pattern could be true for any system artifact.

This problem will be especially acute for innovative domains, where the "cut" made may have little to do with existing architectural partitionings for a given exemplar system. Since ODM is designed to facilitate definition of such domains (although it does not require this) some means is needed to make the functional boundaries of the domain clear before extensive data acquisition is performed.

Approach

The ODM approach to this issue relies on a few key process innovations and accompanying key concepts. The first aspect is reflected in the separation of activities into distinct *Bound Domain* and *Focus Domain* tasks. This obeys the general principle of iterative refinement: Make easier and larger-granularity decisions first, then "hone in." This "incremental focusing" technique serves a risk-reduction role in the definition process. It helps catch big issues early on, encourages

forward progress and guards against thrashing or "boundary vacillation" by creating a hierarchical model for the different levels of defining decisions being made.

The second process aspect is the notion of *settings*. Each exemplar system reflects its own "life cycle" or network of settings which can be related temporally (e.g., development phases) or concurrently (e.g., multiple development sites, multiple user environments). Whereas a particular system will have its particular system settings, it is usually convenient to describe these settings according to a few categories or types. In this Guidebook, we will refer to these generic setting categories as *domain settings*, as distinguished from system settings which would be instances of domain settings. The domain definition will usually encompass only a subset of these domain settings. To add this "domain-in-the-system" view to the previously defined "system-in-the-domain" view, we look at various settings for exemplar systems, considering the artifacts, people and processes involved, and decide which are *relevant* to the domain.

The third and central aspect of this task is the definition of criteria within each selected setting for distinguishing domain-relevant phenomena from those out of scope. In particular, we must learn to distinguish two different kinds of boundary criteria or defining rules for the domain. The first, already reflected in the DOMAIN DEFINING RULES produced in *Bound Domain*, address the domain as a set or class of exemplar systems. The goal of the *Focus Domain* task is to identify, not which systems are in the domain, but which *portion* or "slice" of those systems will be considered part of the domain.

This involves a key distinction. What is the difference between a defining rule as applied in *Bound Domain* and as applied in *Focus Domain*? In the former task defining rules are criteria applied to exemplar systems as a whole to determine whether or not they are domain exemplars. Defining rules in the *Focus Domain* task determine what functionality within an exemplar system is part of the domain's scope of concern. The *Focus Domain* task thus provides a means of systematically out-scoping technical areas within the defined domain boundary for which data acquisition and modeling may not make sense.

> *Example.* In the Outliner domain, text editing functions are intimately interleaved into the operations of creating headings and sub-headings. Yet it would be futile to attempt detailed modeling of all text editing functions as part of domain modeling of outlining capabilities.
>
> However, we cannot consider text editing functions as subjects for a "rule of exclusion" in terms of overall systems, because many outliners we will wish to study are embedded in the broader setting of a text-editing application.
>
> In contrast, systems that support general hypertext links (as opposed to strict hierarchical links) may be excluded from our domain because they have advanced features we want to omit from consideration. The latter is an example of a system feature that results in the system as a whole being labelled a true "counter-exemplar" for the domain.
>
> As an another example, suppose we proposed a rule that the domain boundary exclude all outliner programs that run on workstations, or include all outliner programs that run on Macintosh computers. This would help to bound the domain more precisely. But the system attribute "implementation platform" may not be deemed a feature *relevant* to the Outliner domain itself. On the other hand, a feature like "capability to collapse and expand headings" is a defining feature that directly addresses what makes an outliner an outliner, according to our definition.

As with the *Bound Domain* task, the *Focus Domain* task is primarily concerned with establishing boundaries. Each boundary is "fuzzy" in that we allow for inside, outside, and borderline catego-

ries. Only in the next task, *Situate Domain*, should we begin trying to name or characterize what lies outside the various boundaries identified.

Workproducts

■ DOMAIN SETTINGS

A set of names for particular types of settings that occur in one or more exemplar systems for the domain and which are deemed within the domain scope. The settings are related via links that describe some system life cycle relationship. The domain settings will generally be a subset of all the system settings for the exemplars.

Usually (but not always) this subset will be a contiguous groupings of settings that are upstream or downstream from each other. For example, suppose a a life cycle included the development, customization, field installation and end-user settings. Although the domain has been "bounded" in terms of which systems are exemplars, a domain focusing on development activities would clearly be a "different" domain than one focused on the work processes of the end-users of the system. Either of these could be selected as the domain scope. Or the development and customization settings could be selected as the scope; or the field installation and usage settings. It would, however, be not desirable to focus on the development and usage settings but exclude the customization and field installation settings from the domain scope. This is because one of the purposes of the Model Domain phase will be to model the features of the domain, and to explore ways of shifting certain features across setting boundaries. This means shifting across the intervening settings, and therefore they should be included in the scope of interest.

Note that some system settings will have been identified that are not selected as DOMAIN SETTINGS. These are still worth documenting. Settings immediately "upstream" and "downstream" to the settings chosen as DOMAIN SETTINGS may turn out to be important sources of indirect domain information in the *Model Domain* phase.

■ DOMAIN DEFINING RULES

Within each setting different features can be identified as part of the "domain in the setting" as a whole. Here we use the term "feature" loosely in the text, but codify the features selected in additional defining rules that are added (though kept distinct) from the rules produced in *Bound Domain*.

Just as in *Bound Domain* we had exemplar, counter-exemplar and borderline exemplar systems, in this task we identify individual features as within or external to the domain scope. For each setting in the DOMAIN SETTINGS, we specify the following:

- Domain defining feature. Part of the functional scope of the domain within that setting.

- Setting characteristic. Determines whether the setting is an exemplar for the domain. Defining rules from *Bound Domain* may be refined into these characteristics by being traced to a specific setting, and by being evaluated as external to the domain scope.

- Domain Interface. Features that have ambiguous status. They are allocated to the Domain Interfaces along with criteria pro and con for their possible inclusion or exclusion in the domain scope. These interfaces are analogous to the borderline exemplars identified in BOUND DOMAIN.

When to Start

You can start the *Focus Domain* task as soon as:

- You have some definite exemplars selected and can start investigating which specific settings and features of those systems should be included in the domain focus.

- You have some definite defining rules and can start investigating whether these rules involve domain-relevant features.

Note that neither the EXEMPLAR SYSTEMS SELECTION nor the DOMAIN DEFINING RULES created in the *Bound Domain* task need to be complete for this task to commence. To the extent that there may be iteration and correction of the partial results used, there might be a risk of some wasted work. However, the *Focus Domain* task will also help stabilize earlier decisions.

Inputs

- DOMAIN DEFINITION. The definition as initially developed in the *Bound Domain* task. This includes the following information:

  — EXEMPLAR SYSTEMS SELECTION. Systems of interest that have been designated as exemplars of the domain. This tells us that *some portion* of said systems overlaps the domain's scope. It is the purpose of this task to map out the extent of that overlap in more detail.

  — DOMAIN DEFINING RULES. Principles for why particular systems were included as exemplars. Some of these rules may turn out to have a focusing aspect. Others may be formalize stakeholder-driven criteria.

- SYSTEM INFORMATION. For this task, we will need more detailed access to information about particular exemplar systems: e.g., high-level system architectures, functional profiles, etc. The domain focus will be determined with respect to this information.

Controls

- PROJECT OBJECTIVES. Used to help determine which DOMAIN SETTINGS ought to be considered within the domain focus.

- DOMAIN STAKEHOLDER MODEL. The stakeholders for the domain also have a direct influence on which DOMAIN SETTINGS are selected for areas of focus.

- DOMAIN SELECTION. Characteristics of the domain of focus, determined during the selection process, are used to determine what portions of the exemplar systems belong within the domain scope. To the extent that the domain selection task included technology forecasting (i.e., identifying the "technology window" for the domain), this information is also used in setting the historical context for the domain.

Activities

The primary activities in this task involve refinement of the main outputs from the *Bound Domain* task to a finer level of detail. In the activity descriptions below, we describe how the exemplars are handled first, then the defining rules. This sequence can be varied.

➤ <u>Identify exemplar system settings</u>

Pick a few representative systems from the EXEMPLAR SYSTEMS SELECTION. For each of these systems, examine relevant SYSTEM INFORMATION in order to determine what distinct settings exist for that system. At a minimum, look for development and usage settings, but consider intermediate settings as well, such as maintenance, customization settings, etc. As you identify each setting, add it to a provisional list of candidates from which you will eventually select the DOMAIN SETTINGS.

Look for systems that appear to have some diversity in terms of the settings that occur. In this activity, we are merely identifying the *kinds of* settings that exist for systems in the domain. The object here is to get as complete a list as possible of the distinct settings. We are not yet selecting the settings to include in the domain scope.

> *Example.* In the Outliner domain, most exemplars studied are commercial software packages with embedded outliner functionality. For these, we record a development and usage setting in the list. One exemplar studied is a small software firm that markets an outliner "widget" that can be customized and incorporated into other applications. For this exemplar, we include a "Widget Utilizer" setting.
>
> Two of the text-editor oriented outliners we are considering exemplars handle outliners in different ways. One has a series of outline "level" styles hard-wired into the application. When running in "outline" mode, the application looks for styles of this kind and treats them in a fairly inflexible way. Another outliner, which also uses "level" styles of this kind, provides the user much more flexibility in re-defining the styles and their appearance. Thus, we also include an "end-user presentation customization" setting based on this exemplar.
>
> We do not systematically map every exemplar in terms of these settings. However, for every setting "type" we add to the list, we include a reference to an exemplar that has that setting in its life cycle (as an "existence proof" so to speak).

This activity can be considered complete when no new exemplars considered seem to suggest new types of settings.

➤ <u>Select settings applicable to domain scope</u>

Once we have the list of distinct settings, the next step is to decide which settings are actually relevant to the domain scope. This will require information from the PROJECT OBJECTIVES and DOMAIN STAKEHOLDER MODEL.

The principle used is as follows: We eventually intend to build an ASSET BASE that implements key capabilities of the domain; assets will reorganize and restructure features of certain artifacts, processes and/or stakeholder knowledge in one or more current settings of the domain. Different choices along these lines could effectively result in very different types of domain definitions and different end results.

Identify the setting(s) in which these elements of interest occur. These will be the primary settings of interest for the domain. There will be a natural tendency to consider settings that are immediately upstream and downstream of these settings in the domain-specific life cycle. These can also be considered inside the domain focus, at the discretion of the modelers.

There is a primary trade-off to consider in this activity.

• By choosing a tighter focus with respect to settings, the domain will remain more cohesive

and manageable. There will also tend to be a closer match from existing artifacts and processes as they are partitioned to the eventual assets created.

- On the other hand, picking a broader "neighborhood" of settings as the domain focus allows for possibilities for more significantly transforming artifacts into assets, or shifting functionality across settings. At the most extreme, by considering the entire domain "life cycle" of settings from system development through end-user operations settings (and perhaps beyond, to include end-users as potential customizers and value-added enhancers of the systems) we are effectively doing business process reengineering for the domain-specific software life cycle. The right choice here depends on the PROJECT OBJECTIVES.

Note that later in the life cycle, when we collect data for the domain, we may want to collect supporting data from some settings that are, properly speaking, not themselves within the domain focus. This would constitute contextual data for the domain. Having the DOMAIN SETTINGS clearly established will still be important to keep the data gathering closely linked to the settings where reusable assets will eventually be desired.

> *Example.* In the Outliner domain, the we could distinguish between the setting of "outliner end-user practices" as distinct from "outliner program functionality." If our domain spanned the former setting, we would count as in scope data such as compound operations and sequences of operations performed by end-users (e.g., strategies for collapsing sub-trees, etc.); which might not be part of the latter domain focus. On the other hand, having selected the domain of outliner program functionality as our focus, we might still choose to study end-user processes during data gathering; this would be part of the data acquisition choice, not a definitional choice.

➤ Identify domain defining features for each setting

For each setting selected as a DOMAIN SETTING, identify elements that appear relevant to the domain of focus for that setting. There are a few ways to approach this activity:

- Convert relevant rules. Some DOMAIN DEFINING RULES as developed in the *Bound Domain* task will turn out to have focusing aspects. If so, try to decompose the rules into multiple rules that address the two kinds of criteria separately. This will aid in the tractability of cross-checking the definition later, as well as evolving it over the lifetime of the modeling effort.

> *Example.* In the Outliner Domain, we identified the rule "supports expansion and collapsing of heading displays" as a rule of inclusion for exemplars. We now decide that this is also a defining feature of the outliner functionality within the exemplar systems. This means we simply reallocate the rule as a focusing rule within the DOMAIN DEFINING RULES.

- Some elements of interest, based on PROJECT OBJECTIVES, helped to guide the selection of DOMAIN SETTINGS. These elements can suggest relevant features.

> *Example.* One of the domain stakeholders for the Outliner domain project has indicated that end-users require the capability of selectively searching large, complex documents. Outliner interfaces provide some enhanced search capabilities along these lines. We therefore indicate that in the system usage and development settings, the "search" operation is of interest and should be considered as a candidate for the domain scope.

- Other general features may be identified for each setting through informal information gathering. For examples, work from high-level architectural descriptions if they are available for each system in the EXEMPLAR SYSTEMS SELECTION to identify principal interfaces to domain functionality in the system.

➤ Classify features

Features identified within each domain setting are allocated to one of these categories: Domain-relevant feature, domain-external feature or setting characteristic, or interface/borderline feature.

> *Example.* For the Outliner domain, text editing activities are deemed to be outside the domain. We sort out high-level features that are part of the Outliner domain, part of the text editor, or in the borderline between them. Borderline could mean it falls within outliner structure in one system, in text editor structure in another; or simply that we haven't decided yet.

When To Stop

We can consider the *Focus Domain* task complete when:

- A set of DOMAIN SETTINGS has been selected;

- All previously developed DOMAIN DEFINING RULES have been reviewed in terms of their relevance to domain focus;

- For each setting, a set of domain-relevant features, setting characteristics and borderline features have been determined; and

- All domain-relevant features have been converted into additional DOMAIN DEFINING RULES.

Guidelines

- <u>Don't model differences in features yet</u>. This task may appear to lean heavily on identifying and allocating domain features; yet this doesn't happen "officially" until the *Model Domain* phase. At this point we are looking for high-level features that make a difference in the domain's boundary. Once we decide whether a given feature is in the domain or in a related area, we don't go on to differentiate that feature further. If it falls outside the domain, it is outside our scope; if inside the domain, we'll get to it in the *Model Domain* phase.

5.3.3 Situate Domain

In the previous tasks, *Bound Domain* and *Focus Domain*, planners developed criteria for inclusion and exclusion for the domain, illustrated by exemplar or counter-exemplar systems. In the *Situate Domain* task, planners revisit and qualify the various boundary criteria established for the domain of focus, assign other domain names to bordering areas of functionality, and classify types of relations that hold between the domain of focus and related domains.
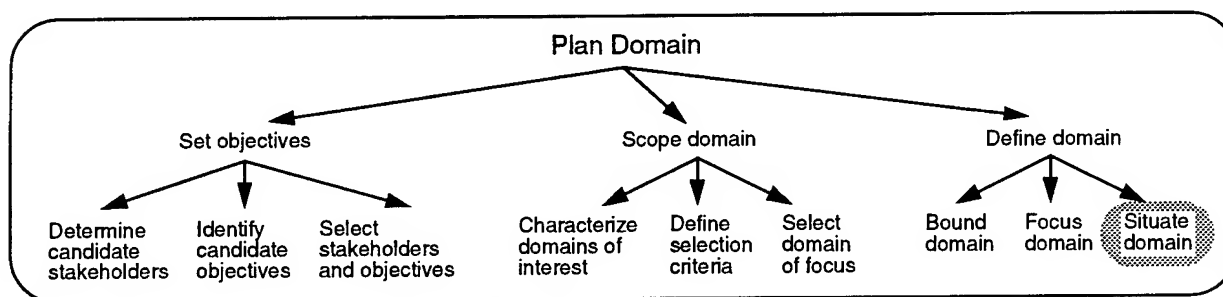
Exhibit 48. Situate Domain Process Tree

A primary goal of this task is to validate and integrate the results of the *Plan Domain* phase as a whole. As the final task in this phase, the *Situate Domain* task serves as a kind of summary for information gathered throughout the phase, and its completion criteria indicate some of the completion criteria for the phase as a whole. In particular, the task validates earlier tasks in the *Define Domain* sub-phase.

A second important goal of this task is to try to further scope the areas that require detailed modeling in the domain. Insights gained as part of *Situate Domain* may help direct attention to neighboring domains. Many of these closely related domains can be approached as separate modeling tasks, and this is in fact often preferable to a large, monolithic model covering too much ground. The vision here is that domain-specific asset bases will eventually be located in a large network of links to other asset bases for other domains. Before doing detailed modeling within the domain, it is helpful to envision the web of these links that form the domain "neighborhood." However, this is an unfamiliar way to think about developing software capabilities, or organizing knowledge resources in general, and thus presents conceptual challenges to the modeler.

Situate Domain also offers an early opportunity for innovation. In checking and validating the formal definition, modelers explore alternative contexts in which domain functionality may be applicable. Where bounding the domain involved a narrowing of focus on a particular domain boundary, orienting the domain involves a complementary widening that mirrors this previous narrowing task while retaining and further refining the domain boundary.

The key challenges in this task are teasing out the remaining contextual assumptions that lurk in our understanding of the domain. The previous two tasks have helped us articulate the assumed boundaries for the domain. But it is still possible to have looked at the domain from a single viewpoint. Clear boundaries will help us design assets for the domain with better separation from context-specific elements. But they will not directly help us to systematically (yet manageably) consider other contexts where domain functionality may be applicable.

Approach

A key to the ODM approach to this task is the specification of a rich set of types of domain relationships. The task focuses on relationships *between* domains, not between systems or system components. This makes it quite different from a conventional systems model or a model of assets in an asset base. As depicted in Exhibit 49, we often intuitively visualize domains as objects connected with links. But this is a misleading image more suited to system components than domains which are conceptual ways of grouping other phenomena. Domains are better thought of in terms of a Venn diagram metaphor. Domains can overlap (i.e., different domain definitions can divide the same world of phenomena up in different and even incompatible ways).

If we slip into the "discrete objects and links" picture of domains, we may imagine that we could identify *all* the domains in a given business area. This doesn't work in the ODM context. The "frame" around the potential domains discovered via such a process is itself an implicit domain boundary. And this boundary will not be understood unless we look outside it as well as at subdivisions within it. The guiding principle is:

We have not understood a boundary until we have considered what is on both sides of it.

Thus the *Situate Domain* process depends on the idea of an already established domain boundary. The situating (or orienting) process involves moving both inside this boundary (to discover significant subsets and components) and outside the boundary (to discover supersets and application contexts). This only makes sense if there is a boundary to move from. In general, activities in *Sit-*

Exhibit 49. Distinction between Arbitrary Domain Relations and Situating a Domain of Focus

uate Domain take bounding and focusing (in or out) decisions and refine them to "name what is outside the domain."

Domain Relations

One major contribution of ODM is the provision of a rich repertoire of these domain relationship types. Many domain analysis methods recommend creation of some kind of domain "context" diagram. What this usually depicts is the domain scope of functionality within a given system, treated as a "black box" and decorated with links to other functional components. Whereas, in a system diagram, the black box usually encompasses the entire system of interest, this sort of domain context diagram can be applied to a subsystem-level domain scope.

ODM significantly extends this kind of context diagram to a much richer set of domain relations or domain interconnections. These relations can be pictured as a set of alternative viewpoints on the domain, as suggested by the picture in Exhibit 49. The typical domain context diagram alluded to above shows up as merely one of the "axes" in this multi-dimensional picture of the domain "situation." The following paragraphs describe the various types of domain relations recognized in ODM. These are not intended to be exhaustive or prescriptive. New kinds of relations may be discovered for any domain, and not all of these will be relevant. But they are a good checklist to work from to help ensure that the domain has been viewed from a variety of perspectives. Taken together, this repertoire of domain relation types have a synergistic effect in producing a robust DOMAIN DEFINITION. They help emphasize the nature of the domain as a shared set of negotiated boundary decisions.

There is no assumption that every related domain identified in this task will already have been formalized as a domain in the full ODM sense, or even that it will be in future. We consider them related *domains* rather than unspecified areas of functionality as a reminder that each of these areas could also be configured in multiple application contexts. However, we believe that in the technology environment of the future, where domains are realized as asset bases on a Web-based network of distributed component brokers, understanding connections between the domain of focus and related domains will become an increasingly critical aspect of domain engineering.

The Domain Neighborhood

A helpful metaphor in situating the domain in relation to other domains is the notion of the "domain neighborhood." (This is not a formal term in ODM, but more of a suggestive image.) The key idea is of a "space" in which the domain of focus floats at the center of things, surrounded, or rather enveloped by other domains.

One moves in the neighborhood not by moving within the domain, but by conceptually shifting a boundary. There are two main ways to move. Starting from the set of defining rules for the domain of focus, one can remove rules or add rules. Removing rules is like throwing ballast from a balloon; the balloon rises and the domain of focus shifts to a *generalization domain*. Adding rules is like adding ballast; gravity and specifics pull you down into a *specialization domain*.

Since a domain is almost always defined with multiple rules, you can ascend and descend at many angles, by removing or adding different constraints on the domain boundaries. Thus the domain of focus can have many generalization domains, and it is a key part of the process in this task that you want to think of *at least three* of them. The reason is that the familiar context with which you have approached the domain will ground you, most likely, in one of these generalizations. (This is part of the implicit context for the domain engineering project, the contextual frame that makes your domain an organization domain and not a pure theoretical construct.) If you think of one more, you may be just *reacting* to the first frame of reference. Think of three, and you are likely to gain insight into the "space" of possible generalizations from the domain of focus. The same technique and rationale apply to looking for specialization domains.

Four principles help bound this conceptual process and keep it from spiralling out of control.

- We have selected the domain of focus. This process won't work unless you are moving out from a focused center point.

- You try to make moves by shifting one rule, feature, etc. at a time. Since there are a finite number of defining rules for the domain, this process eventually (!) terminates.

- Once you make a generalization (or specialization) you don't keep going. You return to the domain of focus to find another related domain. In this way you don't wander off too far; and as the exercise progresses you fill in the neighborhood around the domain of focus more and more thoroughly.

- You also don't consider cross-relations, that is, how one related domain relates to another related domain. You keep everything focused on the (aptly named) domain of focus.

These points are illustrated in Exhibit 49.

The following paragraphs present the primary model relations to be considered in this task.

Domain Classification

By noting set—subset relations in the exemplar sets and/or the defining rules associated with different domains, semantic relationships between these domains can be established. The following specific types of conceptual relations between domains can be documented in the DOMAIN CLASSIFICATION:

- *Specialization domains.* A domain D1 is said to *specialize* a domain D2 if:

  — The exemplar set, or ORGANIZATION CONTEXT, of D1 is a subset of the exemplar set of D2; or

  — The set of defining rules for D1 is a *superset* of the defining features for D2 (i.e., a more tightly constrained definition). In the examples above, "Interactive Outline Browser-Editors" specializes "Interactive Outline Browsers"; and "Army Command and Control Systems" specializes "Military Command and Control Systems".

  Example. For the two domains "Interactive Outline Browsers" and "Interactive Outline

157

Domain of Focus with two "Generalization" domains, D1 and D2

D3, further generalization of D1, would *not* be considered in situating DOF

Relations between D1 and D2 would also *not* be addressed.

Domain of Focus with two "Specialization" domains, D4 and D5

D6, further specialization of D4, would *not* be considered in situating DOF

Relations between D4 and D5 would also *not* be addressed.

Exhibit 50. Relations in the Domain Neighborhood

Browser-Editors", the second would map to a subset of those systems mapped to the first domain. Some outline browsers have editing capabilities, while some (e.g., outline-oriented directory navigation interfaces) do not.

- *Generalization domains.* The **generalization domain** semantic relation is the inverse of the specialization relation.

  Example. The Outliner domain supports features that are a subset of the hypertext domain. Thus an outliner can be considered a "special case" or restricted hypertext system. Similarly, a hierarchical directory browser uses a subset of an outlining directory browser's features.

- *Analogy domains.* In a formal sense, domains D1 and D2 are said to be **analogy domains** if they have overlapping sets of associated exemplar systems or defining features, with neither a complete subset of the other. Analogy domains worth documenting are those that have some pragmatic significance to domain practitioners. An analogy domain may have influenced terminology, design choices, or documented explanations for system behavior provided to users for the domain of focus.

  Example. In the Outliner domain, one analogy domain is revealed in the use of genealogical terms like "parent heading", "child heading", and "sibling heading". (In some versions, "mother/daughter/sister heading" and even operations like "Create aunt" are used.) Here, the domain of human genealogical relations is employed in the system documentation and user interface to provide end-users a familiar, intuitive domain for reference.

- *Organizational Relations.* One way of moving to related domains involves changes in the *assumed organizational scope of applicability* for the domain. (These changes lead to specialization, generalization and analogy domains where the changing rule is in the assumed scope of applicability, not technical features of the domain.) Domains can be associated with the ORGANIZATION CONTEXT in which they are defined. Since a domain involves multiple

exemplars, the ORGANIZATION CONTEXT of the domain of focus may be revealed implicitly in the EXEMPLAR SYSTEMS SELECTION. All exemplars may be systems developed, or used, or maintained within a given organization or division. Consider the set of all possible exemplars that organizational scope. We can now *broaden* that scope by considering a larger organizational entity of which it is a part. Similarly, we can narrow the scope by considering a subdivision of that entity. We are essentially considering supersets and subsets of the set of exemplars characterized in terms of organizational boundaries.

Example. The domain of "Army Command and Control Systems" is a narrower domain than "Military Command and Control Systems". The domain of "Fire Fighters' Command and Control Systems" would be a distinct domain from either of these without a clear subset/ superset relation. But by considering the organization contextual assumptions we are getting a sense of the "organization domain" we are defining, as opposed to an ungrounded abstraction like "Command and Control Systems" in general. Yet we are still not narrowing to the design of a single specific system.

Domain Interactions

- *Component Domains.* These are functional areas within the internal architecture of the domain functional "slice" within exemplar systems. The purpose of documenting these "internal" domain relations is to prevent the later *Domain Model* phase from modeling extraneous system details, perhaps necessary to domain functionality but essentially a separate area of concern.

    Example. In the Outliner domain, most outliner programs have functions to generate a numbered outline header for the document (e.g., 1, 1.1, 1.2, 1.2.1, etc.). This sub-function has its own fairly complex set of options for different styles of enumeration, overriding of defaults, etc. This could be considered a component domain.

    Key Distinction. A specialization domain is generally a restriction of the overall set of defining features for the domain; a component domain is often a localized subset of these features. This is a "kind-of" vs. a "part-of" distinction. In this example, an enumerator is not an "outliner in miniature" in the same way that a to-do list manager is an outliner with many restricted functions.

    Note also that the "outline enumeration" sub-domain might be a component of other applications besides outliners. This is why component sub-domains cannot be treated like components and sub-systems of traditional systems design.

- *Application Settings.* This relation is basically the inverse of the component domain: i.e., we look for settings where the domain of focus serves as a component domain of a broader domain of functionality. This axis is a particularly important one to consider the domain of focus as a component of several diverse application settings.

    Example. In the Outliner domain, some outlining programs are stand-alone, while others are embedded in other applications such as text-editors, presentation managers, and spread-sheet applications. In the context of these other applications, the outlining capabilities function as a component.

- *Peer Domains.* These are functional areas that occur at the same level as domain functionality within exemplar systems. A diagram of domain of focus relations with peer domains would most closely resemble a conventional system context diagram or high-level system architecture. Such a diagram would typically depict the domain of focus as a subsystem, with interfaces illustrated to all related subsystems within exemplar systems for the domain. (This assumes a subsystem-level domain of focus, that is, a horizontal domain relative to the exem-

plar systems.) Peer domains abstract these related subsystems and represent the anticipated range of variability to be encountered in those subsystems across multiple exemplars. These interfaces often raise canonical design issues, e.g., choice of strong versus weak coupling across an operational domain boundary such as a client-server relationship. Major system variants that appear in the domain model may be characterized in large part by design decisions at these boundaries.

Note that unlike a conventional system architectural model, the peer domains relations contains the superset of all major interfaces to domain functions, even where no single exemplar system includes or could include all interfaces simultaneously.

Domain History

- Documents the historical emergence and evolution of the domain as a recognized abstraction in the community of practice. This could involve documentation of predecessor and/or successor domain technologies, forecasts of new technologies that represent a next step beyond the domain's current technology, etc. This historical view is distinct from information about the historical or "genealogical" relations among specific exemplar systems in the domain. This information is gathered in the *Plan Data Acquisition* task that begins the next sub-phase, *Acquire Domain Information*.

Domain Settings

- The software engineering life cycle specific to the organization and domain of focus determines a set of distinct settings for development and use of exemplar systems. The *Focus Domain* task has identified a particular "cluster" of settings that are the focus area for the domain. These settings will be more thoroughly modeled once a specific set of representative systems are selected in the *Plan Data Acquisition* task.

The domain-specific software life cycle for exemplar systems suggests a new set of related domains for the domain of focus. The interactions between work flow across these settings to other settings suggest potentially significant related domains. These are discovered by examining the processes performed and products generated in each setting.

— *Upstream Settings*. These are settings that produce material used as input in the settings of focus. This could include, for example, producers of "enabling technology" for the domain-specific functionality that is the focus of the modeling effort.

— *Downstream Settings*. These would be settings determined to be outside the domain scope but which interact with the focus area for the domain. In these cases the domain capabilities may be serving as enabling technologies for the related domain.

For example, most application domains would have a related test data domain; testing will typically be a function in the development and not the operational environment. Documenting potential usage settings are an important aspect this picture.

Example. In the Outliner domain, several exemplar systems studied build outlining capabilities out of general purpose user interface (UI) capabilities like X-Windows or MOTIF. Clearly, modeling these functional areas should be well beyond the scope of the domain boundary; they are domains unto themselves. Effectively, this is enabling technology for the outliner application. But changes in these general UI capabilities could have a dramatic impact on the outliner implementation; so it is helpful to recognize this as a separate domain.

X-Windows and MOTIF would be, in fact, two separate domains. In one exemplar system, MOTIF might insulate the outliner from direct X-Windows calls; if this were true in all

exemplars, then following the "nearest domain neighbor only" rule for the *Situate Domain* task only MOTIF would be listed as the related domain. (This prevents digging further and further into what might be heavily layered application code; e.g., X-Windows hosted on operating-specific platforms, etc.) However, if other outliner exemplars were implemented directly in X-Windows then it should be included also as a related domain.

Workproducts

The various domain relations described above are partitioned, mostly for tractability, into the separate workproducts listed below.

■ DOMAIN CLASSIFICATION

This workproduct situates the domain of focus with respect to the *conceptual* relations outlined above: i.e., the specialization, generalization, and analogy domains. This workproduct can basically be a simple list of domains conceptually related to the domain of focus. For each entry, annotate the following:

* A provisional name or descriptive phrase for the related domain.

* If applicable, any counter-exemplars for the domain of focus that are allocated to this related domain;

* Indication of the type of relationship with the domain of focus (entries are usually partitioned according to the type of relation, i.e., specialization, generalization, and analogy; so this information is conveyed positionally).

* Any defining rules/features that are allocated to the related domain; that is, the semantic distinction between the related domain and domain of focus, in terms of defining rules. There should be at least one DOMAIN DEFINING RULE for the domain of focus that is "relaxed" in the case of a generalization domain. Conversely, a specialization domain should impose at least one additional defining rule.

  For analogy domains, the relations are a bit more complex. There should be some shared or common defining rules (the basis of the analogy) as well as at least a few contrasting rules or features identified.

* Any borderline exemplars or borderline defining rules that appear to straddle the domain of focus and the related domain.

* Optional: the source or derivation of the related domain; in particular, was the domain a familiar construct for practitioners (an ethnographic or native domain) or is it an innovative partitioning of the functional space?

Exhibit C-9, RELATED DOMAINS/DEFINING RULES MATRIX, contains a worksheet template that can be used to characterize and classify related domains in terms of the DOMAIN DEFINING RULES. Exhibit C-10, CONCEPTUAL/HISTORICAL RELATIONS MATRIX contains a worksheet template that can be used to validate the DOMAIN CLASSIFICATION by mapping domains to their evolution.

■ DOMAIN INTERACTIONS

This workproduct situates the domain of focus with respect to the *structural* relations outlined above: i.e., the component sub-domains, application settings, and peer domains within operational settings.

This workproduct can be structured similarly to the DOMAIN CLASSIFICATION. However, the rules for how defining rules differentiate the domains do not transfer. For a component sub-domain, identify the high-level defining rules or features that are, effectively, decomposed by the sub-domain. In theory, any subset of the domain of focus defining rules could be treated as a sub-domain in this sense, but it only makes sense for certain subsets.

For each key interaction of the domain of focus with a related domain, the interface can be characterized in terms of the original domain attributes, particularly encapsulated vs. diffused. Differentiate structural interactions that are well-contained and modularized (a subsystem interface) from those that involve highly interlocked areas of functionality (e.g., calls to memory management, user interface calls, exception handling).

■ DOMAIN HISTORY

This workproduct situates the domain of focus with respect to *historical* predecessor and anticipated successor systems. It can be formatted similarly to the other workproducts. In this case, look for "transitional features" that help define when a certain technology period ended that differentiates a given domain from a predecessor. Allow for borderline cases as with the other relations. Exhibit C-10, CONCEPTUAL/HISTORICAL RELATIONS MATRIX contains a worksheet template that can be used to validate the DOMAIN HISTORY by mapping domains to their evolution.

Analogical and historical relations may overlap closely. Many automated systems retain the older terminology of superseded manual processes. For example, many computer typesetting systems retain terms and operations from hot-metal printing technology, although the terms often imply a physical mechanism and associated restrictions that is no longer relevant.

When to Start

You can begin *Situate Domain* when:

- A domain of focus has been selected. The task involves a rich set of domain relations. It may be tempting to start documenting these relations for arbitrary domains within the DOMAINS OF INTEREST. However, the coherence of the process described here depends on considering only domains relations with respect to an identified central domain of focus (the selected domain), rather than trying to identify all domain interrelationships within the set of DOMAINS OF INTEREST.

- The domain has been provisionally bounded. Initial DOMAIN DEFINING RULES and EXEMPLAR SYSTEMS SELECTION have been developed. Iteration will be natural between the *Situate Domain* task and the *Bound Domain* task, it is easier to begin situating the domain using initial workproducts developed in *Bound Domain*. The boundary decisions can be used as a basis for discovering

- The domain has been provisionally focused. Initial DOMAIN SETTINGS and DOMAIN DEFINING RULES have been selected in the *Focus Domain* task.

Inputs

- DOMAINS OF INTEREST. A source for possible related domains. Note that DOMAINS OF INTEREST were previously used to determine the DOMAIN SELECTION in the previous sub-phase, *Scope Domain*. In this task we start again from the original list of DOMAINS OF INTEREST to discover related domains. Not all the DOMAINS OF INTEREST will be relevant or related to the domain of focus.

- DOMAIN INFORMATION. Elicited informally in order to identify domain systems and development work settings.

Controls

- PROJECT OBJECTIVES. Identification of related domains for the domain of focus is one strategy to help scope later modeling activities; however, this process itself needs to be filtered in terms of project objectives.

- DOMAIN STAKEHOLDER MODEL. Source and filter for application settings relevant to the domain. Provides guidelines for controlling the level of effort on the task by linking to stakeholder interests and priorities. Related domains identified should, where possible, be of strategic significance within the ORGANIZATION CONTEXT.

- DOMAIN DEFINITION. The definition as refined in the first two tasks of the *Define Domain* sub-phase, *Bound Domain* and *Focus Domain*, becomes a control on this task, a primary objective of which is to stabilize the boundaries by characterizing the neighborhood beyond them. If boundary shifts occur within this task, this constitutes an iteration back to the relevant activities in the earlier tasks.

  - DOMAIN DEFINING RULES. Specify boundary conditions are used to elicit domain relationships.

  - EXEMPLAR SYSTEMS SELECTION. High-level architectural views of exemplar systems are sources for structural or operational domain relations. Documented counter-exemplars and borderline exemplars are sources for conceptual domain relations.

  - DOMAIN SETTINGS. The specification of which system settings (development, usage, maintenance, etc.) are considered part of the domain focus. This is needed for situating the domain because other settings in the system life cycle may suggest significant related domains.

Activities

The following activities employ various complementary techniques to elicit conceptual, structural and historical linkages between the domain of focus and related domains. They are not intended to be performed in strict sequence and not all relations will be relevant to the same level of detail for every domain. The overall flow of activities includes these steps:

- Name the various "neighboring domains" along each axis;

- Validate related domains against exemplars and DOMAIN DEFINING RULES from the previous tasks of *Define Domain*;

- Cross-check various axes with each other; for example, correlate the historical and conceptual dimensions;

163

- Based on the insights gained from the steps above, revisit and validate the various boundary decisions.

The result of these steps is an extremely robust DOMAIN DEFINITION which is the culmination of the entire *Plan Domain* phase and an effective bridge to the *Model Domain* phase. Since PROJECT OBJECTIVES and available resources will rarely allow consideration of all settings relevant to a given domain of focus, new boundary issues that arise that will require iterative refinement of the DOMAIN DEFINING RULES and EXEMPLAR SYSTEMS SELECTION.

The following sections describe the activities in more detail.

➤ Identify candidate related domains

Identify an initial set of candidate domains that appear significantly related to the domain of focus. Record these interim results in the DOMAIN AFFINITY DIAGRAM. Develop the initial list of candidate related domains from the following sources:

- Filter DOMAINS OF INTEREST, identifying that subset of domains connected to the domain of focus via the relation types described in the Approach section above. Not all domains in this set will have relevant relationships to the domain of focus, nor will all related domains appear in this set.

- Interfaces documented in the DOMAIN INTERACTIONS should suggest *structurally* related domains. Find a provisional domain name that encompasses the system functionality in question (e.g., window manager, database system) and document it.

- Working from the DOMAIN STAKEHOLDER MODEL identify application contexts in which domain functionality is developed or used. Elicit related domains to consider from stakeholders, using questions based on the relation types described below.

➤ Elicit related domains from rules

Working from the DOMAIN DEFINING RULES, try successively "masking" one required feature away to see whether a recognizable and distinct related domain emerges.

➤ Characterize counter-exemplars

Working from the counter-exemplars documented in the EXEMPLAR SYSTEMS SELECTION, characterize each counter-exemplar with respect to neighboring domains. Try to find domain names that correlate with the counter-exemplars.

> *Example.* In the Outliner domain, a to-do list manager program may have been considered as a candidate exemplar, then excluded because of the domain defining rule, "Outlining programs can support at least three levels of indentation." Noted as a counter-exemplar, in this activity related domains of "to-do list managers", or "two-level outline browsers", might be identified. Note that these two possibilities

Counter-exemplars identified are not further characterized in relation to the domain of focus within the *Bound Domain* task. Within *Situate Domain*, counter-exemplar systems can be characterized in terms of related domains. Where no related domain can be associated with a counter--exemplar, check the set of related domains is examined for completeness and consistency and add new related domains may be added. For related domains with no counter-exemplars do some additional investigation of candidate exemplar systems.

If traceability is being performed, the results of the validation can be documented in annotations to the EXEMPLAR SYSTEMS SELECTION.

➤ Map related domain affinities

It can be helpful to represent the related domains discovered by means of an "affinity diagram," an informal graphic depiction of the domain of focus in the center of numerous overlapping related domains. This is a useful preliminary way of capturing candidate domains before they are classified more formally. Although placement and conjunction may be used to suggest intuitive relations, the diagram does not attempt to classify the related domains. In fact, it more closely resembles a conceptual association network than a structural system components model. In some cases it may be possible to name the intersection of the related domain with the domain of focus, as in the nature of a Venn diagram. Cross-relationships *between* related domains are not systematically documented in the picture.

➤ Map related domains to defining rules

The RELATED DOMAINS/DEFINING RULES MATRIX can be used to characterize and classify related domains in terms of the DOMAIN DEFINING RULES. A template for this worksheet is illustrated in Exhibit C-9.

➤ Classify related domains

This is the essential activity of the task. Use the descriptions in the Approach subsection of this task for allocating related domains to the appropriate category, and using the category to elicit new related domains as appropriate.

➤ Integrate the relational views

As the final step in this task, synthesize the various views of domain relations, looking for connections between structural, conceptual, historical, and life cycle views.

One suggested technique for this integration is illustrated in the worksheet template shown in Exhibit C-10, CONCEPTUAL/HISTORICAL RELATIONS MATRIX. In this worksheet, domains related conceptually are mapped against a rough time line showing the evolution of the domain. One natural historical trend is for a system to spawn specializations as spin-offs; value-added extensions would be a classic illustration of this. Each extension addresses a *narrower* scope of application, but provides *greater* functionality to support that scope of application. Conversely, a generalization trend reflects developers' discovery that by *removing* certain functions, a system becomes usable in new contexts.

Exhibit 51 presents another approach to the integration of the different views. It is basically a visual aid to help picture the different kinds of "relations in relation" so to speak. Use this view in any of the following ways:

• Map the affinity diagram described earlier (an arbitrary arrangement in space of related domains) onto this positional scheme. Leaving the domain of focus at the center, arrange the related domains according to the axes.

• Follow the diameters across the circle. For example, for a predecessor system try to extrapolate the same movement reflected from the related domain to domain of focus, to a new relation emanating out the other side, possibly leading to a new domain.
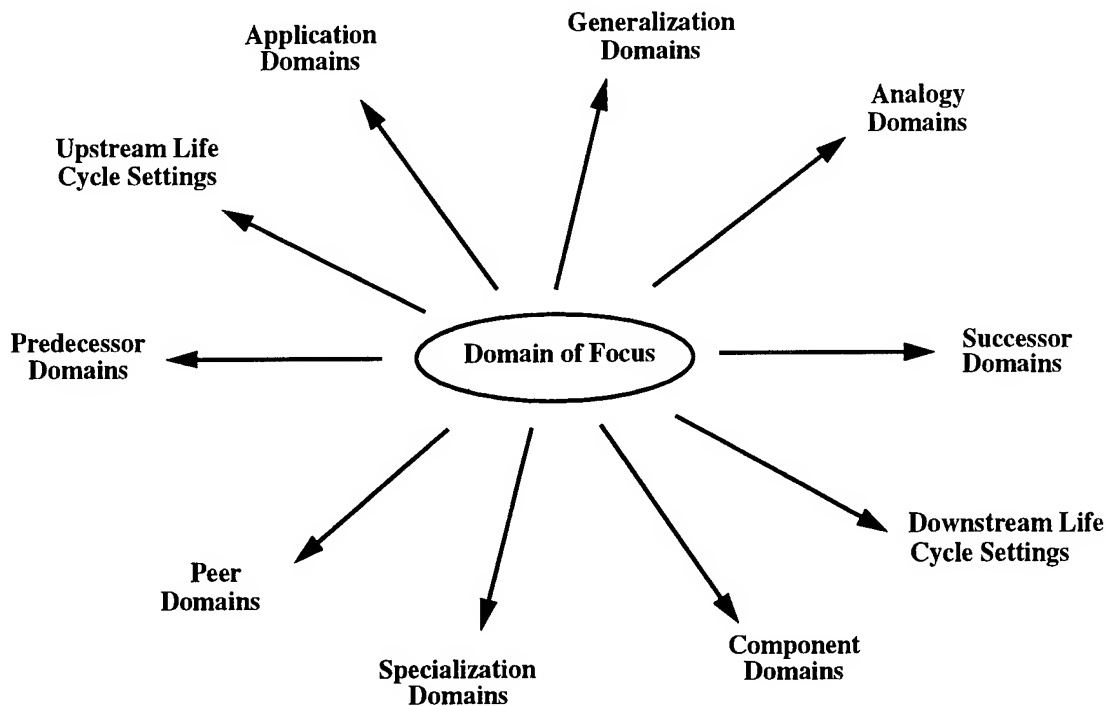
Exhibit 51. View of Domain Relation Axes

- Play with the ordering of the axes, add customized relations and tailor the diagram at will.

➤ Revisit boundary decisions

Based on the steps performed, revisit and validate the domain boundaries as reflected in the DOMAIN DEFINING RULES. Check for the right balance of the following:

- Strategic alignment to PROJECT OBJECTIVES: Now that the domain has "come into focus" more completely, are we focusing on the right things? Have we missed an area that would be considered of essential interest to some key stakeholders? Have we folded in an area that really does not need to be addressed?

- Formal consistency: Are there conflicts in the rules? Are there consistent principles for what is included and excluded in various contexts?

- Ethnographic descriptiveness: does the definition seem to capture the "natural" intuition of domain practitioners. Does the name of the domain reflect this understanding? Are there boundary decisions that seem artificial or contrived, and subject to wide diversity of interpretation?

When To Stop

Assess the domain boundaries that result from *Define Domain* sub-phase with respect to PROJECT CONSTRAINTS (e.g., resources, schedule, staffing, and budget). The relation suggests whether it is appropriate to proceed on to the *Model Domain* phase. The following scenarios indicate the nature of the choices:

- <u>Domain a little too large.</u> You can probably proceed to descriptive modeling. The initial planning step for descriptive modeling will be able to tune or trim down the model coverage in a variety of ways.

- <u>Domain a little too small</u>. Probably the ideal scenario, since descriptive modeling will invariably unearth more detail than you expect. Proceed to detailed modeling. It will always be possible to iterate to improve quality or depth of detail in the model.

- <u>Domain much too large</u>. Iterate the *Plan Domain* phase to define a narrower focus within the selected domain. In order to re-enter the planning loop, the domain-of-focus centered view must be converted to the ORGANIZATION CONTEXT view required as an input to the planning tasks. However, this context will now be documented more explicitly than on the first iteration.

- <u>Domain much too small</u>. The project's process may be broken; i.e., you may have iterated too far. Consider backing out to a wider focus; this will be easier if sufficient traceability and rationale was retained in workproducts to this point.

It is also possible that the process was basically sound, and that the natural domain boundaries fall awkwardly (i.e., last iteration produced a domain of focus too large, but sub-domains are all too small). In this case, several strategies can be applied, including:

- Aggregating several related domains into one descriptive modeling effort (such domains should probably share a primary operational interface, so that the results can be integrated to form a larger functional piece, but also separately reused);

- Changing the project plan to iterate through a series of smaller domain modeling efforts. This approach makes particular sense if learning about and refining the domain engineering process is an explicit objective.

- Spawning several smaller-scale parallel domain modeling efforts within the overall project scope. Select a parallel domain by reentering the *Plan Domain* phase at the *Select Domain of Focus* task, reusing the DOMAINS OF INTEREST report and other workproducts from earlier steps. In parallel proceed with descriptive modeling for this domain.

Guidelines

- <u>Many related domains for each axis</u>. There can be many independent specialization, or generalization domains to the domain of focus. These can overlap or be disjoint in arbitrary ways. Each represents a different viewpoint of what defines the domain of focus as a unique "scope of interest."

- <u>Don't leave the neighborhood</u>. Once you have defined a given generalization from the domain of focus, you need not and should not continue further generalizations. The domain "neighborhood" is formed, as much as possible, by domains related via the smallest possible conceptual "move." This helps keep the process focused; otherwise nothing prevents ranging arbitrarily through various levels of abstraction.

- <u>Triangulate</u>. It is helpful to consider at least two or three alternative related domains for each "axis." Having considered them, though, we do not attempt to model the cross-relationships between related domains defined with respect to the same axis. In a final, validation step of this task, we will look specifically for certain correspondences across axes.

- <u>Follow diameters</u>. You do not have to consider each axis. However, for each axis considered, it is helpful to follow both "directions" of the axis: i.e., generalizations considered together

167

with specializations, predecessor/legacy domains and future trends, etc.

- <u>Elicit via analogies</u>. Probing for analogy domains in interactions with domain informants can be facilitated by a kind of controlled brainstorming that pushes analogies in surprising ways. The process can help to unearth what may have been unconscious design choices and rationale, along with occasional oversights. When an analogy relation is discovered, consider these questions:

  — What features of the analogy domain have been carried over by designers, though not needed in this domain?

  — What features were not transferred, but would be relevant in this domain?

  — What features relevant to this domain have been inadequately addressed because they do not fit the design analogy used?

  This technique can be used particularly effectively by domain analysts who are *not* experts in the selected domain (or who are experts at hiding their expertise).

- <u>Find your own relations</u>. The set of domain relations proposed in this task are a ***starter set*** that is intended to be added to, trimmed, modified as suits the modelers. The most important domain relations to identify may be ones unique to your domain, with no clear axis with which it is associated.

6.0 Model Domain

In the *Plan Domain* phase of the domain engineering life cycle, we have selected a domain of focus based on strategic analysis of stakeholder interests for the project, which ensured a sufficient business case to develop a DOMAIN MODEL for the selected domain. We have also produced a DOMAIN DEFINITION that establishes and delimits clear domain boundaries; by characterizing neighboring areas outside those boundaries, we have anticipated and clarified interactions to be considered in domain capabilities. Our final goal is an ASSET BASE that provides a range of functionality within the domain scope for a specified market of developers.
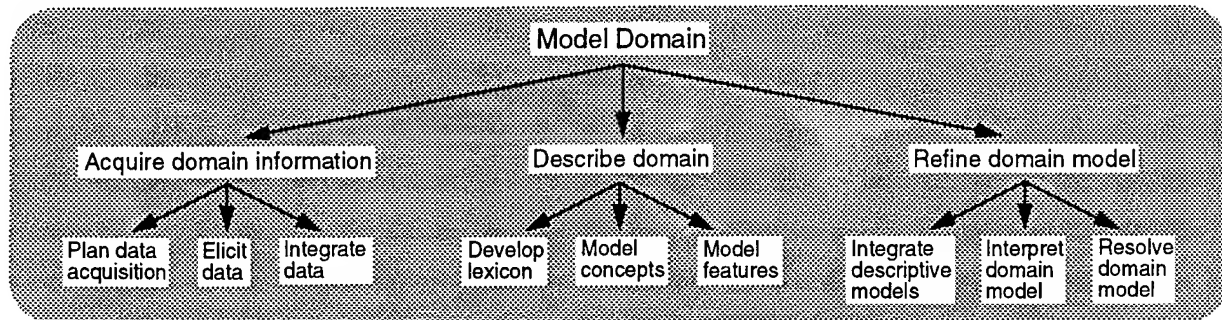
Exhibit 52. Model Domain Process Tree

The purpose of the *Model Domain* phase is to "fill in" the content within the domain itself. This means a shift of attention from *boundary* issues to issues of the structure and conceptual elements *within* the domain. The resulting model should describe the *commonality* and *variability* within the domain. Rather than building a model for a single application in the domain, or even a generic model that may be applicable at a high level to a number of systems, the domain modeling process attempts to formalize the *space of possible alternatives* for individual systems in the domain.

Domain modeling is often compared to knowledge modeling or knowledge engineering in expert systems development. However, when expert systems developers do data acquisition and knowledge modeling, e.g., for a medical diagnosis system, they are usually clear that their expertise in data acquisition and inference systems is distinct from medical practitioners' domain expertise. Furthermore, the task of implementing a knowledge-based system is recognized as quite distinct from a conventional software development task. They understand that knowledge modeling is a clearly distinct task from the primary work of the domain practitioners they work with.

When applying domain modeling techniques in software-intensive domains, the picture can become much fuzzier. It is most often software engineers who are "drafted" into the role of domain analyst. It is tempting for them to fall back on their own engineering knowledge, even when the objective is to capture the knowledge of a different group of practitioners. This is particularly true when the engineers have considerable experience in the domain of focus. Also, in the end, the domain modeling process will aid in constructing software applications of the same kind as those currently in the domain.[1] So it is natural for experienced software engineers to approach domain modeling as a kind of "systems analysis, only more so." Unfortunately, this can create problems, because the domain modeling context lacks most of the built-in constraints that help keeps system modeling efforts manageable in scope. The challenges include the following:

- *Managing data gathering*. To adequately describe and interpret common and variant features

---

[1.] An asset base is quite different from a single application: it might include components for such applications, guidelines for designers, or decision models. But even this notion of an asset base will often be indistinct to engineers at the start of the domain engineering process.

of the domain we need to study exemplar systems more closely and in more detail than was necessary in the *Define Domain* sub-phase. We do not need to build complete descriptions of systems; but we do need comparable data from several systems. This is a resource-intensive task. If we are dealing with a reasonably mature domain (one of the general criteria for a good candidate domain) it is a good rule of thumb that there is almost always too much data. Knowing how much data to collect, from what sources, and to what degree of detail, is an ongoing challenge for the domain modeler.

- *Scoping modeling effort.* We must scope not only the data to be acquired but the models to be derived from that data. A system model is complete when a system can be implemented from the model. A domain model has no such clear criteria for completeness. How do you know when you're done? How much detail is sufficient? Deriving appropriate exit criteria for the modeling process is a key issue in domain engineering.

- *Respecting domain terminology.* Modelers naturally impose their own terminology on the models they develop, especially as they discover the need for differentiating concepts that arise through comparative analysis. Since modelers are often not the primary practitioners in the domain, their modeling decisions may not reflect the language or concerns of the practitioners who will eventually need to own, maintain and use the models.

  This presents a fundamental dilemma for domain modeling. How do you use "native" domain terminology appropriately, while avoiding the assumptions that come along with the terminology?

- *Controlling modelers' bias.* A related challenge involves taking into account the modelers' bias. Cognitive processes like memory, analogy, and classification are intrinsic elements in modeling. Modelers' insights into the significant commonality and variability implicit in domain information are shaped by such factors as their relative experience in the domain, the number of contrasting examples seen, the order in which they are examined, and the delay between examining data and developing models. The model is a product of the data selected for examination, the representation strategies and the particular process followed. How can the domain modeling process acknowledge and help manage these factors?

Approach

The ODM approach to domain modeling reflects a related set of principles that work together to address the above challenges. These principles, further elaborated in the following paragraphs, include the following:

- Separation of system and domain modeling;

- Empirical grounding of models in specific *representative systems*;

- Maintaining a focus on *descriptive modeling*; and

- Explicit modeling of variability within the domain.

Preserving a clear boundary between system and domain modeling activities is essential for the coherence of the domain modeling process. This helps modelers distinguish when they are examining system workproducts or interviewing practitioners as a domain modeling activity, versus as a system-building activity. This also allows the process to be applied to settings where different software development paradigms are followed, or to non-code workproducts and processes of the software life cycle, or even to non-software domains.

Exhibit 53. Model Domain IDEF$_0$ Diagram
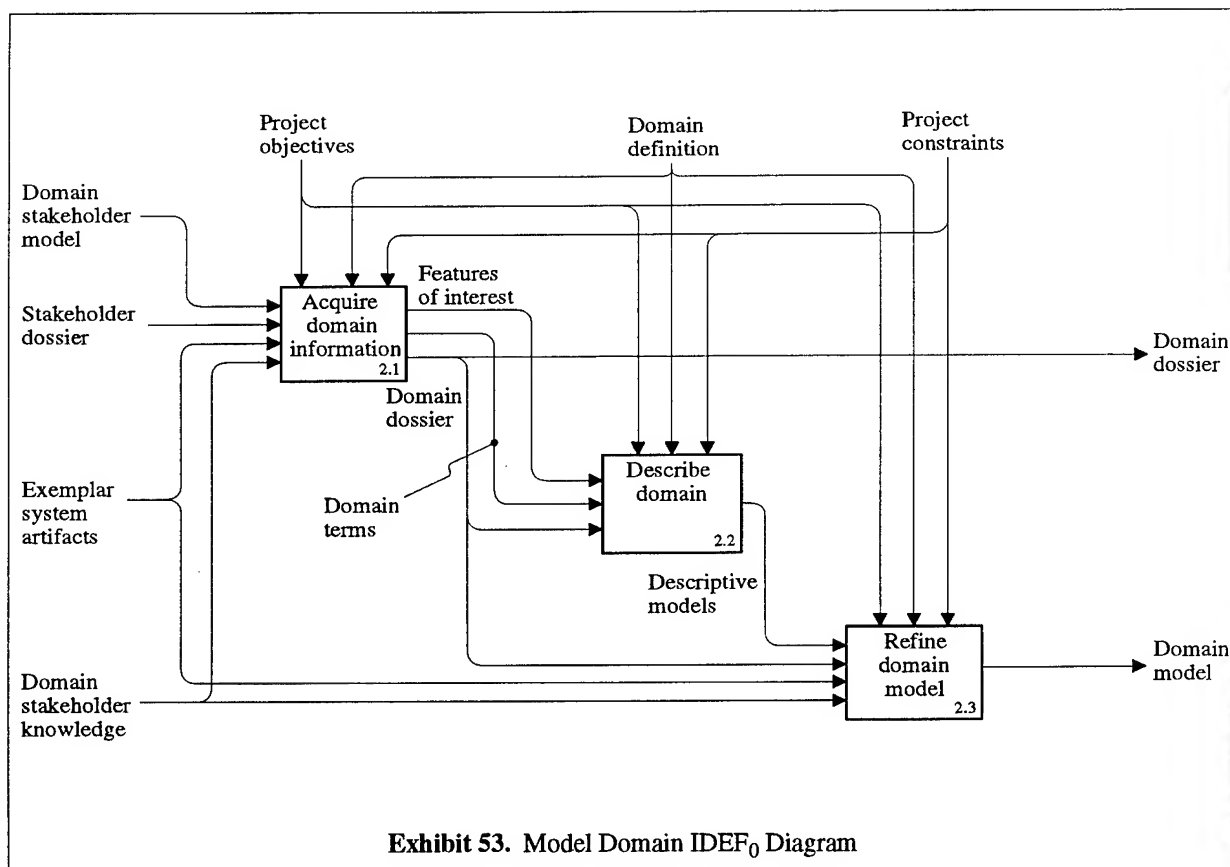
In ODM, domain models are *empirically grounded*. To set boundaries for how much variability to incorporate, modeling is based on information acquired about a specified set of exemplar systems called the REPRESENTATIVE SYSTEMS SELECTION. Although we have studied examples before this point, we now plan the data to be examined more systematically, to make sure that we get a representative sample of the variability within the domain, given our resources and constraints. The range of variability described in the DOMAIN MODEL is validated with respect to this explicit representative set. The rule for what goes into the DOMAIN MODEL is as follows:

- Anything described in the DOMAIN MODEL must have *precedent* in an exemplar; and

- All representatives must be adequately described in the DOMAIN MODEL.

The ability to trace a model to the specific data used in developing the model helps document the context for the models, making them more usable and robust for future developers. The REPRESENTATIVE SYSTEMS SELECTION thus gives us a way of both knowing when the DOMAIN MODEL is "done" and clarifying to ourselves and others what "done" means for this model. The use of multiple representatives also helps adjust for the bias imposed by abstracting from one example, while recording the entire set of data used provides a "map" of the data that influenced the model.

ODM focuses the *Model Domain* phase on *descriptive* modeling. When we confront differences in the domain, one variant may seem more intuitively satisfying or familiar than another, so we will often be tempted to "choose sides." However, if both the variants are within the domain scope, then we want to describe the variants impartially. We are not making deciding how to build a single application in the domain; we are mapping the kinds of decisions that need to be made.

171

This suggests why the *Model Domain* phase of ODM involves explicit modeling of domain variability. Some approaches to domain engineering assume a DOMAIN MODEL includes *only* common features within the domain. In ODM the DOMAIN MODEL includes *both* features that span the variability observed across the REPRESENTATIVE SYSTEMS SELECTION.

> *Example.* In the *Plan Domain* phase for the Outliner domain, we clarified the boundaries of the Outliner domain and informally studied several exemplar systems. We now have a pretty good idea of the kinds of systems we want to consider in and out of the domain; for example, outliners in file system browsers and text editors are in, outline capabilities in the rows and columns of table-based tools like spreadsheets are out.
>
> In the *Model Domain* phase we are going to select a few outliners to study in detail; these will be our REPRESENTATIVE SYSTEMS SELECTION. As we study different outliners, we discover different styles of outliner behavior, different strategies for implementing outliners, different kinds of work processes where outliners can be used. For example, one style of outliner allows headings to "jump levels" somewhat arbitrarily, while other outliners use a strictly hierarchical notion of what an outline is. We will document the range of variability in outliner applications in the DOMAIN MODEL. If we discover new defining rules that characterize all outliners, these will be documented in updates to the DOMAIN DEFINITION.

Results

The DOMAIN MODEL, the primary workproduct created in this phase, describes common and variant features of systems within the domain, and rationale for these variations. When performing the *Engineer Asset Base* phase, the DOMAIN MODEL will serve as a basis for selecting a range of variability to be supported by the ASSET BASE.

Note that the DOMAIN MODEL can include design and implementation features. Some approaches to domain modeling assume that a DOMAIN MODEL includes only high-level system requirements or functional requirements. While the DOMAIN MODEL in ODM does as a rule include such requirements, it can also include low-level requirements, performance requirements, and design and implementation features. In principle, the DOMAIN MODEL can describe commonality and range of variability for any process or workproduct in any phase of the domain-specific system life cycle.

A secondary product of this phase is the DOMAIN DOSSIER which documents the specific *information sources* used as a basis for modeling. The DOMAIN DOSSIER is a valuable output of domain engineering because it serves to organize and consolidate many sources of domain expertise and experience in a coherent form. The DOMAIN DOSSIER is also used during the *Engineer Asset Base* phase to trace legacy artifacts that are candidates for reengineering into reusable assets, and to identify constraints in systems into which assets will be migrated.

Process

As shown in Exhibit 53 the *Model Domain* phase consists of three main sub-phases:

- The *Acquire Domain Information* sub-phase involves filtering relevant domain information, e.g., from *system artifacts* or from interviews with *domain informants*. Acquired information is retained in the DOMAIN DOSSIER. The DOMAIN TERMS used by domain practitioners are also documented.

- The *Describe Domain* sub-phase involves the core activities that analyze domain data to produce models of the common and variant concepts and features in the domain. In practice, a

number of separate DESCRIPTIVE MODELS are developed addressing various aspects of the domain and various levels of abstraction, including rationale about specific alternatives and trade-offs within individual systems. The process acknowledges the fact that we will generally not know all the models needed or their formal interconnection at the start of modeling.

- The *Refine Domain Model* sub-phase involves integrating separate DESCRIPTIVE MODELS into a single integrated DOMAIN MODEL. This model includes interpretive data about the rationale for the commonality and variability observed in systems in the domain. In addition, the model can include *innovative features* suggested during previous modeling activities.

These distinct sub-phases help modelers distinguish subtle transitions in the modeling process that are often a source of errors. For example, when data acquisition and modeling are freely intermixed there are dual risks of either of generating too much data without a comparative framework or jumping into modeling so quickly that insufficient data is considered. When the purely descriptive modeling activity and the refinement activities of integration, interpretation, and innovation are interwoven without distinction, it becomes difficult to define exit criteria for the process as a whole, or to assess which aspects of the models reflect design decisions imposed, sometimes unwittingly, by domain modelers.

Guidelines

There are many different "life cycles" woven together into the activities of this phase. There is an overall planning, data gathering and analysis, and reflection phase which may iterate multiple times, enriching the data set with each pass. Each primary information source also undergoes a process of analysis and interpretation and may contribute to multiple, closely interrelated models. In addition, each model integrates data acquired from multiple information sources.

The transformation of information can be performed in broad stages as suggested by the sub-phases, or more incrementally and at a finer level of granularity. The sub-phases of the *Model Domain* phase can thus be performed in a number of different sequences. The sequence chosen will have a dramatic impact on the structure and content of the resulting DOMAIN MODEL. Several general "styles" of doing the modeling can be identified:

- A *concept-driven* modeling style works from the concepts, and focuses on acquiring data to populate a specific model. In this approach, modelers use a selected *concept area* as a filtering control, and sift domain information sources for data relevant to that concept area. Concept areas of interest thus inform decisions about what data to collect, e.g., what information is extracted and traced from artifacts (via keyword search, highlighting, etc.), what questions are asked in interviews, etc.

  One advantage of this approach is that it is easier to control acquiring necessary and sufficient data for each model. One disadvantage is that the same information source may be analyzed numerous times by different modelers with possible redundant effort. If the information is derived from a human *informant*, it may not be possible to have multiple interviews, and therefore the DOMAIN DOSSIER will need to be carefully maintained.

- In an *information-driven* modeling style, modelers begin with a specific information source and attempt to filter relevant data from that source into multiple appropriate models. One advantage of this approach is that information sources can be very thoroughly modeled, and may suggest new and important concept areas not originally planned. Disadvantages include difficulty in regulating the appropriate level of detail to model and keeping in scope. (The DOMAIN DEFINITION and REPRESENTATIVE SYSTEMS SELECTION are important mechanisms in this regard.)

173

Various hybrid strategies are also possible. One possibility is a matrixed team structure that involves each modeler in both data acquisition and modeling activities. Each modeler is given responsibility for both a set of information sources (perhaps including particular informants as well as system artifacts and documents for particular representative systems) and particular concept areas. Each modeler distributes data acquired from their information sources to other modelers, and receives data in turn for their own models. This approach requires a high degree of team coordination and *team modeling* skills. (See Section 9.3 for a discussion of team modeling techniques as an optional layer of ODM.)

6.1 Acquire Domain Information

In the *Plan Domain* phase we have selected and defined a domain of focus for the project. The clear boundary conditions established in the DOMAIN DEFINITION provide a basis for acquiring detailed information about the domain from which to develop the DOMAIN MODEL.
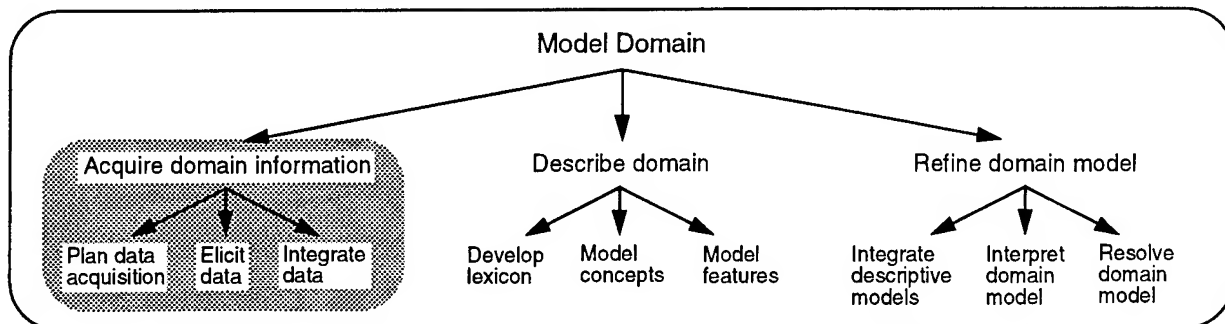
Exhibit 54. Acquire Domain Information Process Tree

The primary purpose of the *Acquire Domain Information* sub-phase is to systematically gather information on which to base the DOMAIN MODEL,

The *Acquire Domain Information* sub-phase addresses a number of challenges. A representative set of systems must be selected that will provide appropriate information for the comparative modeling that will be done later in *Describe Domain*. For all of these systems, context recovery poses particular challenges of managing the biases of the ***modeling team***, monitoring availability and quality of the data, and controlling the settings for which the data is collected.

An additional challenge is that the information from other contexts must be made explicit, not merely internalized by the modelers. Although domain modelers will learn a lot about the domain in the *Acquire Domain Information* sub-phase, this alone is not enough; the information must be made available for later sub-phases of modeling and for asset base engineering, and possibly for hand-off to other project team members as well. If modelers are familiar with the domain they must find a way of making their own tacit knowledge more explicit and transferable; if modelers are unfamiliar with the domain they must learn a new area (get their "sea-legs") rapidly and still codify the knowledge elicited in a form amenable for modeling.

Approach

Information about the domain is obtained by examining ***information sources***, including any system ***artifacts***, which can be any system workproduct used as a source of information, as well as human ***informants***, who can provide information about a system. The artifacts can include documents about a system, code, demos, specifications, etc. The informants should be taken from a number of settings, i.e., users, developers, maintainers, administrators, etc. The information sources will be examined by a team of ***investigators***; their activities will include interviewing of humans, and analysis of and experimentation with artifacts. A key aspect of the ODM approach is to perform data acquisition in a way that facilitates ***context recovery*** (see Section 3.4 "Key Aspects of the ODM Method") on a selected set of artifacts, to make information embedded within them explicit.

Exhibit 55. Acquire Domain Information IDEF$_0$ Diagram
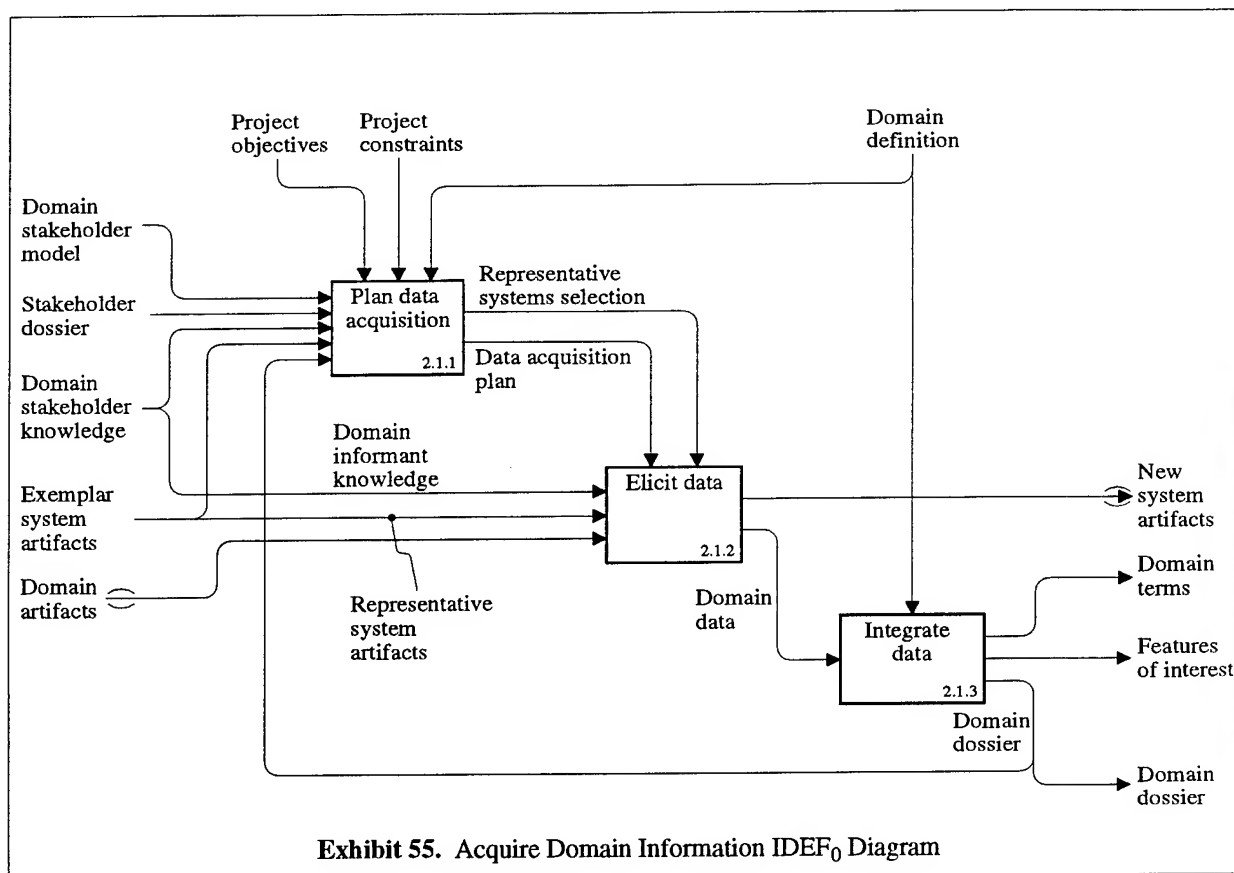
Results

The primary result of the *Acquire Domain Information* sub-phase is the DOMAIN DOSSIER, which is a coherent record of the information gathered from all the selected information sources. This provides the raw materials for the subsequent *Describe Domain* sub-phase. Since the descriptive models will be about the selected domain, the dossier should be limited to information about systems in the selected domain, as specified in the DOMAIN DEFINITION. Since the domain model will cover variability between systems, the dossier should include overlapping information about several systems.

As shown in Exhibit 55, there are three primary tasks in this sub-phase. These three tasks constitute a cycle of information gathering in which the information required is determined, gathered, and integrated into the dossier, from which further requirements can be deduced. These tasks are:

- The *Plan Data Acquisition* task involves all aspects of planning for data acquisition, ranging from scoping the data to be examined, deciding where the information will come from, who will gather it and how, down to determining particular questions that need to be answered. Whenever you are making a decision about whether or how to acquire a particular piece of data, that is, to ask a particular question, to examine a particular document, to study a particular system, you are doing *Plan Data Acquisition*. Any choices that need to be communicated to other team members or kept for later reference are documented in the DATA ACQUISITION PLAN.

- The *Elicit Data* task focuses on actually answering questions that were determined in the *Plan Data Acquisition* task. This can be as simple as asking a question to a user or looking up

a question in a manual, or as complex as designing and carrying out an experiment on a running system.

- The *Integrate Data* task involves relating the data gained in *Elicit Data* to other data acquired so far. This can be as simple as noting the answer given to a question, or as complex as integrating the information gained from an interview into a task-flow diagram of the use of a system. Constructing categories based on comparative observations of multiple systems is properly the job of *Model Domain*, and does not belong here in *Integrate Data*. The coherent, integrated corpus of knowledge produced here forms the DOMAIN DOSSIER.

All three of these tasks are best performed by the guidelines drawn from the Information Acquisition supporting method area. Because of the tight interaction among the three tasks, a supporting method in this area needs to be able to provide support for all three simultaneously, that is, it needs to be able to provide guidance for planning, elicitation and integration. This supporting method area is described in Section 8.2.

Guidelines

- <u>Iteration between planning, elicitation and integration</u>. It is unrealistic to imagine that one can select all information sources that need to be examined before actually examining any of them, or to plan all sessions with a particular source before carrying out a single one, or even to know all questions for which one might want to find answers from a single source, before finding the answer to a single one. The examination of some information source can suggest further information sources to examine, or further information to be gained from the same information source. Even a single question answered by an information source could indicate further questions that need to be asked (just as looking up a word in a dictionary often prompts us to look up another word). Therefore, the *Plan Data Acquisition* task of *Acquire Domain Information* is a continuous activity performed concurrently with *Elicit Data* and *Integrate Data*.

- <u>Planned versus opportunistic data gathering</u>. There is a trade-off to deciding how to explore emerging data possibilities. One extreme is to determine beforehand what information needs to be gathered, and stick to it as closely as possible. Another extreme is to 'let the data lead', and be reactive to newly found sources of information. The most useful policy usually lies somewhere between the two. A key design challenge in this process is balancing the need for completeness in data acquisition with the inefficiency of examining redundant or irrelevant data.

6.1.1 Plan Data Acquisition

Included in the final output of the *Plan Domain* phase was the DOMAIN DEFINITION, which defines membership in the domain. In *Plan Data Acquisition*, we determine what information will be acquired about the domain, where we will get it, who will collect it and how.

The key challenge in *Plan Data Acquisition* is to make certain that the information obtained adequately recovers information from the settings relevant to the domain. This challenge occurs at many levels of detail; from identifying the setting in which to examine a system, to planning how to avoid imposing an alien structure on an informant's knowledge. The details of how to address this challenge at various levels is left to the appropriate Information Acquisition supporting method. In the overall context of the domain engineering life cycle, the *Plan Data Acquisition* task provides:

- documentation of the empirical data that forms a basis for evaluating the completeness and
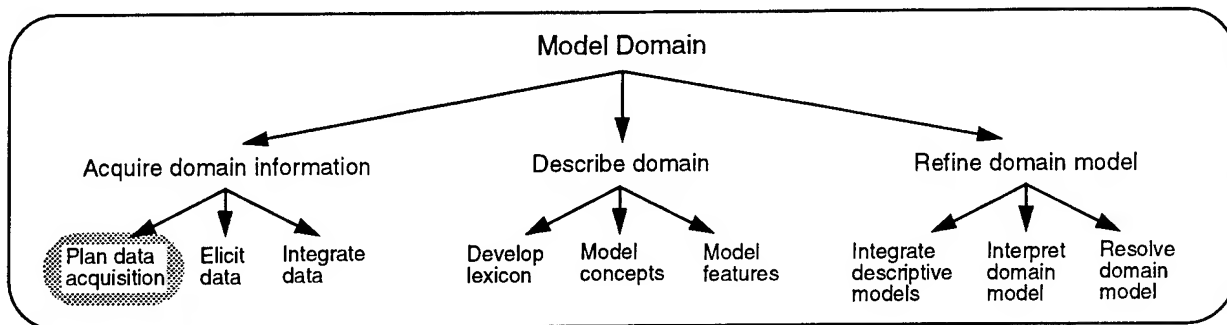
Exhibit 56. Plan Data Acquisition Process Tree

consistency of the DOMAIN MODEL to be produced; and

- a strategic framework for what data will be gathered about the domain, based on PROJECT CONSTRAINTS and characteristics of available domain information sources.

Approach

It is not possible to study every known system that is relevant to some domain; therefore, during *Plan Data Acquisition* we select a subset of known domains which should be representative of the commonality and variability throughout the domain. It is necessary to determine the specific subset of the EXEMPLAR SYSTEMS SELECTION to study as *representative systems*. A number of issues arise in addressing the challenge of selecting the set of representative systems that adequately cover the variability in the domain. The choice of representative systems is not simply a statistical sampling of the systems; special considerations must be made for the fact that the goal of domain modeling is the comparative analysis of systems. In selecting the information sources to study for each representative system, it is necessary to consider the range of *system settings* for which information is available.

Once the representative systems been chosen and their system settings determined, the problem of recovering information from the various settings of those systems comes to the fore. A major issue here is the bias of the investigators. It is important to remember that the modelers come from their own work culture, and have their own point of view on the domain. This is particularly important when doing domain modeling for software reuse, since the modelers themselves will probably have experience in software, and will bring their own cultural biases into the session.

As investigators carry out data acquisition, their background knowledge will change. The order in which a particular investigator examines basic texts, tutorials, system artifacts and technical documents, or interacts with domain practitioners will influence the data that is acquired at each stage. It is, therefore, important to consider the development of an investigator during the *Plan Data Acquisition* task.

Workproducts

- **REPRESENTATIVE SYSTEMS SELECTION**

  - A list of systems, annotated with rationale behind why they were considered to be a representative set for the purposes of domain engineering. The REPRESENTATIVE SYSTEMS SELECTION includes information about the system settings to be studied for each representative system. Each setting is related to one of the DOMAIN SETTINGS determined during the *Define Domain* sub-phase of the *Plan Domain* phase. A mapping of representative systems against DOMAIN SETTINGS. Exhibit C-11, REPRESENTATIVE SYSTEMS INFORMATION QUALITY MATRIX, shows

a worksheet template that can be used to create the mapping from representative systems to DOMAIN SETTINGS. The mapping is used to select system settings to examine.

■ DATA ACQUISITION PLAN

A plan detailing information sources to be investigated for the domain. It includes the following:

• Data acquisition goals, which state what information is required from each information source, in sufficient detail to guide data elicitation. Typical goals include discovering native terminology, determining limitations of operations and filling in steps of procedures. Goals can also contain information about the scope of the investigation, e.g., limiting the investigation to certain features of a system. Data acquisition goals are usually hierarchical (including high-level goals for the entire investigation, down to low-level goals for a particular session), and may be written in outline form. A goal should indicate which system or set of systems from the REPRESENTATIVE SYSTEMS SELECTION and which information sources from the list in the DOMAIN DOSSIER are to be investigated.

• A list of all information sources for the systems in the REPRESENTATIVE SYSTEMS SELECTION. Information sources include artifacts (including user documentation, manuals, executable code, source code, design documents, presentations, tutorial materials, etc.) and informants. Each information source should be annotated with the system setting for which it can provide information.

When to Start

• The domain has been selected. If this task begins before domain selection the risk is that excessive resources will be applied in characterizing information availability for all systems within the ORGANIZATION CONTEXT (i.e., more than just exemplar systems for the selected domain). Such an inventory might have value to the organization but goes beyond the needs of domain engineering.

• The domain has been bounded. The *Bound Domain* task has produced an initial DOMAIN DESCRIPTION and EXEMPLAR SYSTEMS SELECTION and they have been cross-validated for consistency. If the process begins before the domain boundary is reasonably stable, there will be insufficient criteria for selecting representative systems or relevant information types.

• The domain has been focused. The *Focus Domain* task has produced an initial DOMAIN SETTINGS to help determine which settings are of interest for data acquisition.

It is not necessary to have determined all DOMAIN INTERACTIONS before this task begins.

Inputs

• DOMAIN STAKEHOLDER MODEL. Each category of stakeholders is assessed in terms of:

    — the value of the stakeholders as informants; and

    — what can they tell us about the domain.

    For example, distinctions could be made between novice, journeyman and expert experience levels in the domain.

• STAKEHOLDER DOSSIER. The STAKEHOLDER DOSSIER contains supplemental information about stakeholders. This is a primary source of information about potential domain infor-

mants for the information sources listed in the DOMAIN DOSSIER.

- DOMAIN DOSSIER. The evolving "map" of the data sources acquired so far. At the beginning of planning, it will be empty, but after a few rounds of Integrate Data, it will contain structured domain data. Further planning will make use of this data to determine what further data needs to be elicited.

  A particular entry in the DOMAIN DOSSIER is concerned with genealogy information about the representative systems. This provides a record of all influences one representative system has had on others, including subsequent versions and competitors. The genealogy information is updated throughout the *Acquire Data Acquisition* sub-phase, just like the DOMAIN DOSSIER in general.

- EXEMPLAR SYSTEM ARTIFACTS. Artifacts available for each exemplar system. These artifacts are inventoried and classified by type of artifact.

- DOMAIN STAKEHOLDER KNOWLEDGE. Other artifacts and informants available for the domain are inventoried. These include secondary data sources (survey articles, textbooks, etc.) that are not associated with particular exemplar systems.

Controls

- PROJECT OBJECTIVES. Both overall objectives and domain-dependent objectives refined during the *Select Domain of Focus* task provide a basis for filtering and prioritizing the data to be acquired.

- PROJECT CONSTRAINTS. Project resources will assist in determining how many representative systems can feasibly be studied and what artifacts will be feasible to examine given the experience and skill level of the domain engineering team staff.

  If PROJECT CONSTRAINTS include specific staffing and scheduling information, this information may flow into the DATA ACQUISITION PLAN in the form of specific team assignments for interviewing and artifact analysis.

- DOMAIN DEFINITION. The EXEMPLAR SYSTEMS SELECTION component, which documents all systems considered during *Define Domain*, is the source for the REPRESENTATIVE SYSTEMS SELECTION. The DOMAIN SETTINGS help determine which representative system settings will be of use in data acquisition.

Activities

Planning of data acquisition in ODM involves the challenges of managing the various settings in which the representative systems will be studied, as well as managing the biases of the investigators. Many of the activities below are aimed at these ODM-specific goals. All of them should be considered in light of the Information Acquisition supporting method.

All these activities are interdependent, and can depend on results of the *Elicit Data* and *Integrate Data* tasks as described in the guidelines section of *Acquire Domain Information* above.

➤ Select Representative Systems

Adequate description and interpretation of common and variant features of the domain requires more detailed data from exemplar systems than was necessary in the *Define Domain* sub-phase. However, this could be a resource-intensive task. If we are dealing with a reasonably mature

domain (one of the general criteria for a good candidate domain), then a good rule of thumb is the following:

There is always too much data at the start of analysis — until specific questions arise, then there is never enough data.

One way to control effort expended in data acquisition is to restrict the number of systems we will study. Picking a specific set of representatives is critical to establishing exit criteria for the phase as a whole. The rule for what goes into the DOMAIN MODEL is as follows:

- Anything described in the DOMAIN MODEL must have some precedent in an exemplar, and

- All representatives must be adequately described in the DOMAIN MODEL.

The REPRESENTATIVE SYSTEMS SELECTION thus gives us a way of both knowing when the DOMAIN MODEL is "done" and clarifying to ourselves and others what "done" means for this model.

To control the effort expended in the *Plan Data Acquisition* task, filter the EXEMPLAR SYSTEMS SETTINGS to form the REPRESENTATIVE SYSTEMS SELECTION. The filtering should take into account the PROJECT OBJECTIVES, including domain-specific objectives. Some general criteria to consider in selecting the REPRESENTATIVE SYSTEMS SELECTION are:

- Potential factors to consider when selecting individual representative systems include:

  — *Data Coverage.* Relative availability of data for each candidate representative system. Are there enough different information types to research for a given candidate representative system?

  — *Position of the exemplar system within the EXEMPLAR SYSTEMS SELECTION.* Is the system core or close to the borderline of the domain boundary?

  — *Data Accessibility.* How easy will it be to gain access to information for this system?

  — *Data Quality.* What is the overall quality of the system as an example to study?

- Potential factors to consider for the set of representative systems as a whole include:

  — *Stretch.* Inclusion of corner cases and a wide range of diversity. One type of diversity is structural diversity (e.g., *domain functionality* includes both encapsulated and diffused implementations).

  — *Finesse.* Select at least a few systems that are functionally very similar. This will ensure that the DOMAIN MODEL will be rich enough to express subtle variations in domain functionality as well as coarse-grained variation.

  — *Comparability.* For a given information type, is there a rich enough cross-section of data available from the different representatives selected? For example, can we study requirements documents from at least three systems?

  — *Completeness of coverage.* Does the set of candidates as a whole provide adequate coverage across the desired range of data?

Below is an example of taking these factors into consideration.

Example. In the Outliner domain, the Macintosh System 7 Finder™ was chosen as a repre-

sentative system because of its ubiquity, and the availability of a number of expert users as informants. The outliner in Microsoft Word™ was chosen to contrast outliners used for document processing with those used for file organization. A 'corner case' of WebArranger™ was included to cover the case in which the outliner is used to manage a distributed data situation. The emacs outliner was included since source code was available. In general, the systems that were selected are commercially available ones, in accord with the project objective to cover familiar functionality.

➤ Document genealogy of candidates

Include in the DATA ACQUISITION PLAN goals to determine and document the historical inter-relationships and dependencies among candidate representative systems. Documenting these historical relationships, at least at a large-grained level, helps make a REPRESENTATIVE SYSTEM SELECTION with sufficient diversity of dependent versus independent development. Later in the modeling process (particularly during the *Interpret Domain Model* process) this historical information will help modelers to interpret common and variant features that were observed across representative systems. Genealogical information gathered both in this task (to support planning) and later in the *Elicit Data* and *Integrate Data* tasks should be recorded in the DOMAIN DOSSIER.

Useful genealogical relations to document include the following:

- *direct successor.* System A superseded system B, adding new features, correcting errors, and observing constraints on compatibility.

- *leveraged.* Artifact (code, and/or design, tests, etc.) from system A was adapted for use in system B, possibly to implement the system for a new customer.

- *requirements reuse.* Operational features of system A were adopted for system B (typical for software products of competitors who begin "feature wars").

- *independently developed.* Systems were implemented in different contexts, and without reference to each other (e.g., applications developed within separate organizations).

- *competitively developed.* Systems were developed for the same target environment (e.g., shadow projects, redundant systems development for reliability, competitive designs to published requirements, applications developed in educational settings).

The set of representatives should include as wide a cross-section of these relations as possible, with particular emphasis placed on independently developed systems. Commonality observed across such systems is likely to represent core functionality for the domain. Avoid selecting a set of representatives that are all members from one systems family. However, intentionally selecting two systems of close lineage as part of the representatives may yield valuable comparative data.

➤ Scope representative systems

Using the scoping and focusing decisions recorded in the DOMAIN INTERACTIONS, scope the intersection of domain functionality with each representative system. The domain of focus may be vertical with respect to some representative systems, and horizontal with respect to others, and may be encapsulated or diffused within different representative systems. One system may be a stand-alone implementation of domain functionality, another including it as a discrete component, function or subsystem, a third implementing a diffused subset of functionality. The scoping information from this activity can be used to constrain the data acquisition goals below.

Example. Outliner functionality appears in a large variety of systems, including word proces-

sors, personal data managers and file system organizers. Each of these brings with it a suite of other functionalities, including searching, displaying, modifying and sorting, all of which interact with some outlining functionality. For each representative system, it will be necessary to decide which functionality is within the scope of the Outliner domain, and which is not. In the DOMAIN INTERACTIONS, it was decided that different display modes for the data in a single grouping would not be included in the Outliner domain; therefore, when planning to read documentation for WebArranger, which offers a large suite of such features, this part of the documentation was not included in the goals for that session.

➤ Select system settings to study

A unique challenge in data acquisition for domain modeling is to realize that the systems that are being studied appear in several different settings. The concept of settings in ODM was introduced in Section 3.4. In order to provide a thorough coverage of the domain, it is not sufficient just to study an appropriate range of systems, but it is also necessary to study these systems in a wide range of *work practice settings*. In this activity, the settings in which the systems in the REPRE-SENTATIVE SYSTEMS SELECTION appear are determined and documented.

Beginning with the DOMAIN SETTINGS, determine how each representative system appears in these settings; the manifestation of a domain setting in a particular system is called a *system setting*. As illustrated in Exhibit 57, a single exemplar system may have multiple settings of any of these types.

> *Example.* Suppose that DOMAIN SETTINGS include two settings, development and usage. In the Outliner domain, the outliner in Microsoft Word™ has a single development setting, within the programming team that develops the product. It also has a number of *usage settings*, for example, collaborating book authors, students, technical writers, etc. These are all system settings of Microsoft Word.
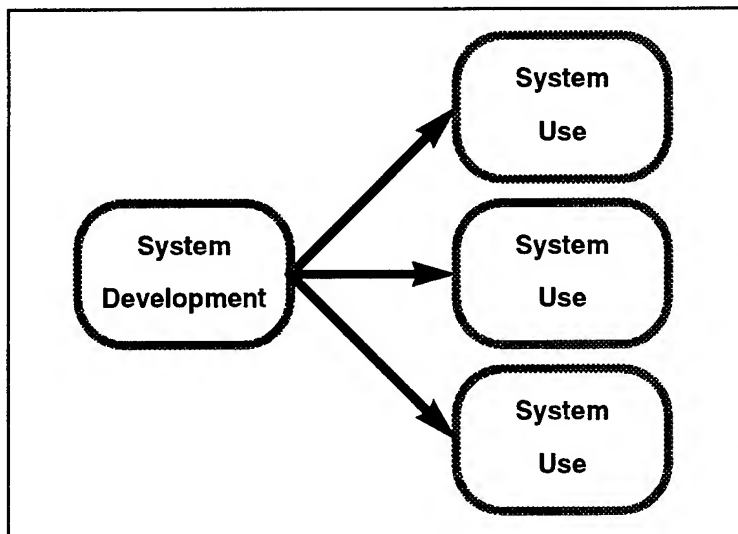
Exhibit 57. Typical Exemplar System Settings

During data acquisition, some system settings might be discovered that do not correspond to any known DOMAIN SETTINGS. In such a case, a new setting can be created and recorded in DOMAIN SETTINGS. Keep in mind the constraints on DOMAIN SETTINGS from the *Define Domain* sub-phase when adding new settings.

Example (cont). Microsoft Word™ also allows for a certain degree of customization of the outline format, by allowing a user to specify indentation levels for the various sub-structures, and other indicators of level (such as face and font). These preferences can be saved on their own, independent of the outline itself. This means that there is the possibility of another setting between developer and end user, where someone designs and creates outline formats.

Record all system settings for each system, along with the domain setting to which it corresponds, as systems settings information in the REPRESENTATIVE SYSTEMS SELECTION.

Map all systems in the REPRESENTATIVE SYSTEMS SELECTION against the DOMAIN SETTINGS. Exhibit C-11, REPRESENTATIVE SYSTEMS INFORMATION QUALITY MATRIX, shows a worksheet template that can be used for this activity. Each cell corresponds to a single system, and a single domain setting. If there is no system setting corresponding to that system and domain setting, then the cell is filled in with N/A. Otherwise, summarize the availability of data in all the system settings corresponding to that domain setting, based on overall quality and availability.

Based upon this information, as well as the PROJECT OBJECTIVES, select which system settings will be of interest for data acquisition. Record these decisions in the DATA ACQUISITION PLAN.

Example (cont). We have identified a single development setting for Microsoft Word. But, since it is a commercial system, development information is not available. On the other hand, we have identified several usage settings; some of these might be easily accessible and others difficult. We summarize the overall accessibility and quality of all usage setting in the table. We record these facts in the REPRESENTATIVE SYSTEMS INFORMATION QUALITY MATRIX, and decide not to consider the development setting of Microsoft Word as a target for data acquisition.

➤ Catalogue information sources

For each system in each setting, record all available information sources in the information sources listed in the DOMAIN DOSSIER. Keep in mind that information can come from a variety of sources, including:

- Artifacts (primary data, workproducts generated and used by domain practitioners)

- Secondary data (surveys, historical analyses, industry information, consumer reports, textbooks)

- Practitioners (developers, expert users, casual users, help-desk operators)

- Work processes (how practitioners use systems and artifacts in their work)

Information about informants can be found in the STAKEHOLDER DOSSIER and the DOMAIN STAKEHOLDER MODEL. This catalog will grow as new information sources are uncovered during the course of data acquisition.

Example (cont). Information sources for Microsoft Word™ available to this project include the User's Guide, the executable code for the Macintosh, and the on-line documentation. The Windows executable is also available, but is not installed anywhere available to the domain modeling team. Furthermore, there is one expert user in the workplace who uses the system on a regular basis to produce documents, as well as other users, who make more limited use of the system. Finally, the workplace often receives documents in Word format, from which some part has to be printed. The (undocumented) process by which this happens can be an invaluable source of information about system. All of these sources, along with the settings

in which they are relevant, are recorded in the DOMAIN DOSSIER.

➤ Characterize biases of investigators

If an investigator enters an interview or analysis session with a particular idea of how a domain is broken down into different areas (either borrowed from his own background, or acquired from some other domain setting), then this risks contamination of the session, preventing the recovery of information from its own work setting. The trade-off here is that while domain-naive investigators bring fewer biases to the session (clean slate), they risk appearing unprepared to an informant, or being unable to understand the subtleties of an artifact (haven't done their homework).

Biases of investigators performing domain acquisition cannot be eliminated, but they can be controlled and accounted for. Characterizing each investigator's bias, and tracking them as they develop, can provide the information needed to control the data acquisition process. The key is to realize that biases are not weaknesses on the part of a particular investigator; they are inevitable aspects of human interaction. Everything you learn changes how you see things. There is an advantage both to sophistication and to naivete; but sophistication is easier to gain than naivete is to regain.

- *Evaluate investigator's technical background* with respect to representative systems/settings. Technical sophistication in an investigator is a two-edged sword; a technically naive investigator will have difficulty evaluating a technical document or understanding a discussion with a technically sophisticated informant. On the other hand, a technically naive investigator will have fewer biases based on technical issues, and might be able to see the data from a broader perspective. Keep in mind that an investigator's technical sophistication will change from one investigation to the next.

- *Evaluate investigator's background with respect to the project*. During the course of data elicitation, certain directions are agreed upon by the modelers and the informants. For example, a project-specific terminology might be agreed upon, to simplify certain communication tasks. It is important to know how familiar a particular investigator is with this terminology. An investigator who is new to the team cannot be expected to have the same assumptions as investigators who have already committed a great deal of time to the acquisition process.

- *Evaluate investigator's bias about organization of information*. Each investigator should make explicit his biases about what constitutes well-organized information for a particular session. Some common biases include conditions on consistency or completeness of the data, ambiguity of the terminology, or even details of the representation of data, e.g., as a time-line or a hierarchy. Mismatches between information sources and investigators on this issue can be a great cause of frustration.

- *Make subjective biases explicit*. Each investigator should keep track of any other biases he discovers that influence the way he views the data. It is, of course, difficult to make hidden biases explicit before an interview or artifact analysis, but the mere act of writing down biases as they are discovered, either before or after the fact, can help clarify the role of biases in the data acquisition enterprise.

Below is an example of investigator bias.

Example. In the Outliner domain, Julie is a member of the data acquisition team who has been exposed to outliners as a user, but has no experience in other settings. The data acquisition team has come up with a lexicon of terms to disambiguate operations that, in various systems, go by the names *hide, conceal, collapse, expand, reveal, show* and so on, but Julie has not yet studied this lexicon. She assumes that a user interface will, at least superficially,

appear object-oriented, where various data types have sets of operations that are somehow 'orthogonal' wherever possible. Finally, she assumes that an outliner will be integrated into whatever system it outlines.

➤ Assign investigators to sources

Based upon biases, availability of investigators and information sources, and skills of the investigator, plan an investigation session. Some investigators might be particularly skillful at interviewing informants, others at reading design documents, still others at running code and experimenting. As an investigator's biases and skills develop throughout the data acquisition activity, the criteria for making this choice will change.

> Example (cont). Julie's biases, along with the fact that she has considerable experience with the World Wide Web (WWW), both in user and installation settings, makes her an excellent choice to examine the demo of WebArranger, an outliner system connected to the WWW. Since she is not familiar with the shared lexicon, she will concentrate on determining the WebArranger-native terminology for its features. Her bias toward orthogonal representations of system functionality works to her benefit in ensuring coverage of all of WebArranger's features. In fact, this bias was the result of an explicit decision during *Plan Data Acquisition*. The final bias, that an outliner should be part of the system it outlines, was revealed as she examined the WebArranger demo, and she discovered that the outliner is separate from the WWW Client. In future investigations, she will not have this bias, or at least, not so strongly, having seen a counter example.

➤ Determine data acquisition goals

Even for a single system in a particular setting, the range of information that could conceivably be gathered is enormous, and must be bounded if data acquisition is to end.

Goals for data acquisition appear at several different levels and are interdependent; goals for studying a particular system will constrain the goals for a particular session concerning that system, and the goals for the session will impact the goals for a particular line of questioning or experimentation. In general, a goal at one level is set by examining data acquired so far, pursuant to some higher level goal, and determining what information is still missing to achieve the higher level goal. Methods for structuring acquired data, and thereby determining what data are missing, are properly part of the Information Acquisition supporting method; they range from interviewing techniques such as 'grand tour' questions to dossier management techniques that can illustrate what systems have been examined so far.

> Example (cont). The highest-level goal in the data acquisition project is to cover all the systems mentioned in the REPRESENTATIVE SYSTEMS SELECTION. WebArranger was chosen as a representative system because of its unique relationship to the WWW; therefore a goal was set in the study of WebArranger to explore its features especially as they related to the WWW. During the investigation of WebArranger functionality, it was discovered that WebArranger links to the WWW via a particular 'note type'. The high-level goal of understanding the WebArranger's relationship to the WWW, along with the gathered information about the URL note type, generated the new knowledge acquisition goal to learn about note types and their role in WebArranger.

The highest-level goal is always to cover all the representative systems sufficiently to support modeling commonality and variability during descriptive modeling. This means that a rudimentary form of comparison between the systems will be necessary, to determine if all systems have been investigated to corresponding levels of detail. Once data is available to perform a detailed

comparison, then it is time to move into descriptive modeling, and systematically model the domain.

> *Example.* The highest-level goal of *Acquire Domain Information* is to cover the domain in preparation for descriptive modeling. The Macintosh System 7 Finder has a structure called a *folder* that can contain other structures, including other folders. WebArranger also has a structure called a *folder*, which plays a roughly analogous role. In anticipation of determining commonality and variability of these systems during descriptive modeling, it will be necessary to elicit information about these analogous features to similar levels of detail. In this case, the analogy was easy to find, and was even reflected in the names of the structures. In general, finding analogous features is more challenging.

Data acquisition goals will normally indicate the system(s) to be studied in what settings, who the target audience will be, and what aspect of the system to focus on. Record the goals in the DATA ACQUISITION PLAN.

➤ Arrange elicitation logistics

A number of logistic problems will arise in the planning of data elicitation; the details of what logistics are needed are highly dependent on choice of an Information Acquisition supporting method. Some activities might be irrelevant for a particular data elicitation session. It is essential to keep in mind the issues of bias mentioned above while carrying out this logistic planning.

- Select the guidelines for the form of data acquisition that will be employed for each information source; this can also help to set further data elicitation goals as mentioned above.

- Form the interviewing teams.

- Arrange access to the information source. This may involve issues such as cooperation among organizational divisions or separate organizations, funding for the informants time spent in interviews and meetings, data classification and security, etc.

- Arrange equipment needed for the investigation. In the case of human informants, pay special attention to the effect that recording equipment might have on how the informant responds.

When to stop

The initial data acquisition planning task can be considered complete when:

- The data acquisition effort has been scoped in terms of representative systems, specific system settings and types of information sources to be studied.

- Representative systems have been selected. The selection has been validated in terms of providing an adequate data sample, with adequate access to quality

- All data acquisition goals have been accomplished. The domain acquisition effort is finished when its plan says that it is finished; this means that all the goals (including the topmost level goal) have been completed.

Guidelines

- Find a domain confidante. Although the purpose of *Plan Data Acquisition* is to be systematic about what sources of information are utilized in domain modeling, it is necessary to establish some starting point. In practice, it's useful to identify a *domain confidante* who can tell

you what artifacts are worth looking at, who is considered an expert within the domain, what other informants to talk to, etc. In some cases, the confidante will also be a sponsor who can facilitate the access to the information sources, but the roles can be distinct as well.

- <u>Representative systems are not necessarily customer settings</u>. There are good reasons to study systems that may not be candidate receptor systems for implemented assets. Back-filling components into deployed legacy systems that have been in use for some time may be quite difficult. Requirements to back-fill might constrain implemented assets in ways that would frustrate their use in other, more critical contexts. Nevertheless, study of these systems may be quite valuable in deriving a complete and robust descriptive model. By separating selection of systems as information sources from systems as potential customer settings, the ODM process encourages modelers to examine a rich diversity of data about the domain.

- <u>Stay flexible</u>. Information sources become available at indeterminate times (especially in the case of human informants). Remember that any planned strategy is provisional at best.

- <u>Be aware of sources of conflict</u>. Data acquisition is, above all, a communication exercise, and anything that can be a barrier to communication can be a barrier here. Of particular importance in data acquisition is the blindness caused by conflicting world views; be prepared to let go of some assumptions that you might consider quite valuable.

6.1.2 Elicit Data

In the *Plan Data Acquisition* task, a goal for knowledge acquisition was set, an information source chosen, and an investigator selected to acquire some information. The investigator has been prepared as well as possible for the session, taking into account background and biases. However, even with the best-laid plans, information can still remain hidden or be misunderstood. The main challenge in *Elicit Data* is to make sure that appropriate communication lines are maintained that will allow the data acquisition goal to be accomplished.
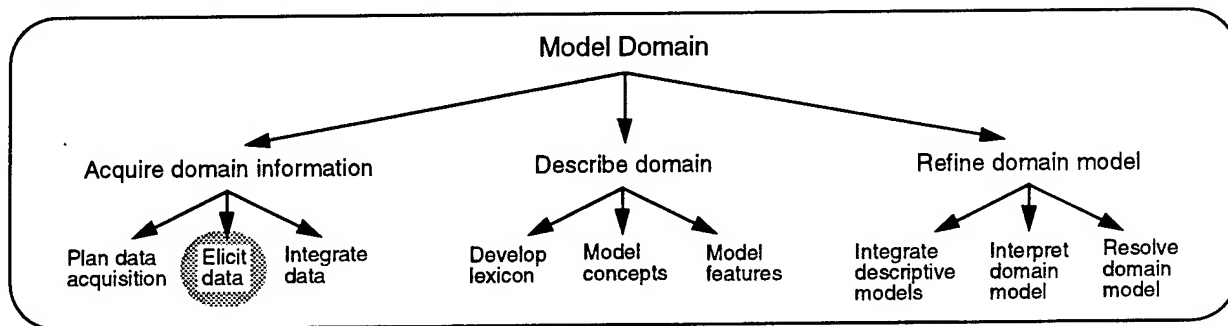
Exhibit 58. Elicit Data Process Tree

Approach

The ODM approach to data elicitation is a responsive one; while direct methods (direct inquiry or consultation) are the simplest to apply and are effective in a vast majority of situations, it is always imperative to watch for possibilities for miscommunication. Having a plan that identifies biases and *work practice settings* is invaluable input to such an effort, but using this preparation to manage communication lines requires the particularly subjective skill of being able to see things from a point of view that is, by design, not one's own. Activities in this task focus on the issues of accounting for the different viewpoint of the investigator.

Artifacts

So far we have talked about exemplar systems, some of which have been selected as representative systems for data acquisition. These systems are created and used in a series of system settings. In each setting, material and information is created and transferred from person to person to get work done. These are system workproducts, and it is common to consider all the workproducts created throughout the software system life cycle, from requirements through design, to test plans, code and user manuals. Any and all of these system work products can become a source of domain information. When used in this way we term the work product an artifact. In the ODM context, an *artifact* can be broadly defined as a system workproduct created by a domain practitioners (as opposed to a member of the domain engineering team), used as a source of information for domain engineering.

The term artifact denotes the fixing of information in actual documentary form (such as on-line textual information). Domain information (e.g., knowledge about the domain in an informant's head or expressed in a conversation) must be fixed into the form of some artifact (an interview report or transcription), before it can be dealt with concretely in domain engineering. At this point it becomes an artifact.

In most cases using a workproduct as an artifact for domain engineering means interpreting it in ways quite different than its original purpose within its own setting. A requirements in the setting where the required system is being built is an *imperative statement*. Viewed as an artifact by a domain engineer, a requirement is data, data about the settings in which the requirement was written an in which it will be read. In particular, the single requirement may be analyzed in combination with several other, similar requirements, where the object of data gathering will be to gather descriptive data about the *range of requirements* of different exemplars in the domain.

It is helpful to distinguish different kinds of artifacts, based on how directly we can correlate the artifact with a specific representative system. Recall that we want to understand how domain capabilities vary across the representative systems; so it will be important to keep straight what data came from what artifacts and which artifacts are associates with which systems. Artifacts include:

- *Primary artifacts* which are used as workproducts in creating exemplar systems, including tools. Primary artifacts can be directly correlated to a single exemplar system. A typical example would be code, or a design or requirements document that specifically addresses a given system. This would be contrasted to a secondary artifact like a survey article discussing several systems or talking about general functionality in an application area. Primary artifacts are also the subject of direct comparative modeling during the *Describe Domain* sub-phase, and are typically candidates for reengineering into assets.

- *Secondary artifacts*, including review articles, commentary, surveys etc. Secondary artifacts can include exemplar-specific artifacts as well as artifacts associated with particular informants but not with one particular exemplar system. Secondary artifacts generally have less clear correlation to a single exemplar system.

Other artifacts to be considered can include *legacy models*, informal or formal domain modeling efforts by domain stakeholders (earlier domain models, architectures, etc.) These must be handled differently from other artifacts because they contain direct taxonomic modeling information. For example, a *generic architecture* developed by a design team using informal reuse techniques could not be easily compared with a representative system built for a single usage setting. Also, artifacts generated by the domain engineering team itself (e.g., via reverse engineering) should be distinguished from primary artifacts.

Informants

So far our project has involved organization stakeholders from the ORGANIZATION CONTEXT, some of whom became project stakeholders and, after domain selection, domain stakeholders. *Domain practitioners* are people who are involved in domain settings. For software domains, these practitioners can include developers, testers, field installers, customer service reps, value-added resellers, maintainers, and, last but not least, end-users of systems. We use the term practitioner to emphasize that people situated in work settings of any kind are not just agents performing processes, but people who share culture, language, habits, previous experience and tacit knowledge with their co-workers. A network of such people is sometimes called a *community of practice*.

As part of the *Elicit Data* task investigators interact with certain practitioners to obtain domain information. Practitioners involved in these interactions are termed *informants*, (borrowing a term in common usage in the social sciences) i.e., people from whom information is obtained that contributes to the domain engineering project. We use the term informant rather than expert because much valuable information is derived from people who do not consider themselves experts.

Only selected practitioners are used as informants, and only selected system workproducts are used as artifacts in the context of domain engineering.

Workproducts

■ DOMAIN DATA

Data elicitation is basically finding out the answer to (a set of) questions; organizing these answers, writing them up, placing them in a comparative structure is properly part of *Integrate Data*. The hand-off from *Elicit Data* to *Integrate Data* is done through the relatively unstructured means of recording DOMAIN DATA, including:

- video and audio recordings made during an interview or observation session,

- handwritten notes made by the investigator during an analysis, or

- laboratory notebooks kept while collecting data during an experiment or an automated observation.

■ NEW SYSTEM ARTIFACTS

Any workproducts for representative systems that are prototyped or reverse engineered to fill gaps in the found data. Typically these workproducts will have some other use, but would not have been created if not for their additional value as sources of domain information. Because they are created as data for domain engineering, rather than to build any individual systems, they are considered *artifacts* rather than workproducts. Because these artifacts are typically associated with a single representative system, rather than models of the full domain scope, they are termed "*system* artifacts." Because the artifacts are created in the course of domain engineering, rather than being pre-existing artifacts examined by domain engineers they are "*new* system artifacts."

When to Start

Data elicitation can start as soon as any data acquisition goals have been set in the DATA ACQUISITION PLAN. Eliciting data without a specific goal will risk collecting data that is not relevant to the selected domain or project objectives.

Inputs

- DOMAIN INFORMANT KNOWLEDGE. Knowledge elicited from domain practitioners in interviews or via observation and process capture. Knowledge can include knowledge about specific representative systems or more general knowledge about domain concepts, terminology, etc.

- REPRESENTATIVE SYSTEM ARTIFACTS. System artifacts to be studied (according to the plan)

- DOMAIN ARTIFACTS. Sources of information about the domain not associated with a specific representative system; i.e., not a system workproduct.

Controls

- DATA ACQUISITION PLAN. The relevant part of the plan is the data acquisition goals, which contain enough information (system, information source, investigator, etc.) to drive the elicitation.

- REPRESENTATIVE SYSTEMS SELECTION. A list of which systems will be investigated; the plan goals should only refer to systems in this list.

Activities

The activities below are best considered a repertoire of alternative techniques rather than a fixed sequence of activities. Recall that the overall goal of data acquisition is to make information that initially is only available in some work settings accessible to the domain modeling team. Techniques for accomplishing this range from very direct to highly interpretive. Selection among these methods depends heavily upon the details of the particular piece of data and the work setting from which it will be drawn. Any data acquisition project of reasonable size will draw on many of these.

➤ Interview informants

Human informants pose a particular set of challenges to data elicitation based upon psychological, sociological, and other human factors. While details of any interview should be taken from the Information Acquisition supporting method, the following considerations can help guide an interview:

- Interview styles (question sets, loose versus tight facilitation) appropriate for different practitioners, e.g., field technicians, system administrators, expert designers.

- Choice of location, on-site (close to the informant's work setting) versus off-site.

- Interviewing individuals separately, versus groups of practitioners. For group interviews, drawing together cross-functional groups from the same project, or practitioners who play analogous roles on different projects.

- Possible interview protocols to use:

  — Scenario-based elicitation of work process (talking through a task);

  — walk-throughs/inspections of artifacts in interview settings;

  — soliciting "war stories" about past project experience; or

  — requesting direct comparative evaluations from expert informants (e.g., people with experience on multiple systems).

➤ Prepare artifacts for comparative analysis

Some aspects of data elicitation apply primarily to inanimate artifacts. Different representative systems will be at varying stages of the software life cycle at the time that domain engineering is performed. For the purposes of comparison, however, it may be important to have workproducts from multiple systems reflecting the same life cycle phase. These activities result in NEW SYSTEM ARTIFACTS.

This activity involves using supporting methods in the following areas, as documented in Section 8.0:

  — Reverse Engineering

  — System Modeling

  — Process Modeling

Depending on the state of the artifacts, some of the following steps might be necessary to get the data into a state suitable for analysis:

- *Reverse engineer to fill gaps in available data.* For example, if architectural diagrams of all representative systems are desired, some may need to be generated from detailed design documents. This activity can draw on Reverse Engineering supporting methods.

- *Prototype artifacts.* This is the converse of reverse engineering. For example, if only requirements were available for a given system, a prototype architecture diagram could be generated for the purposes of analysis. This would involve, in effect, doing the necessary bit of system development to get the data created. However, this might typically be created as a prototype only if being created primarily to produce comparative data for domain engineering. (If the project is coordinated with a parallel system development effort, it is debatable whether this parallel effort could be practically considered as taking place under the umbrella of the domain engineering project.)

- *Translate or convert artifacts into common representations/notations.* These may also be required for the purposes of comparative modeling. For example, if several different design notations were used for different systems it will be much more difficult to compare them for common and variant objects and operations. Some common representation might greatly facilitate comparison.

  Both the activities above would make use of whatever System Modeling supporting methods were appropriate for the software development practices within the organization.

- *Modeling the work process in development settings.* If development settings are within the scope of the domain and the DATA ACQUISITION PLAN, it may be appropriate to develop mod-

els using Process Modeling supporting methods appropriate to the organization. (These are considered distinct from System Modeling techniques in that they have been applied to describing workflow in development settings as well as usage settings.)

Some strong caveats apply:

- Perform the activities above only when conditions require it. In general, this can be resource-intensive work, so caution is advisable. In particular, generate artifacts for representative systems *only where necessary* for domain engineering objectives.

- If artifacts are developed by project members who are not also members of the application teams for the representative systems in question, they should *not* be considered primary artifacts. That is, they are not data created directly by domain practitioners, and therefore might represent the various agendas and technical biases of the modeling team.

➤ Observe practitioners in their settings

Direct observation of work practice in the domain can be a powerful elicitation technique. Social scientists are familiar with what is known as the "say versus do" dilemma. What people report that they do often bears faint resemblance to what they do in practice. Try to directly observe the core work practices that characterize the domain.

For usage settings, this could include watching system operators perform domain functions using fielded systems. One purpose for observation of system operators would be to discover needed domain operations that are performed in a usage setting, e.g., work-arounds to system bugs or limitations, aggregate series of operations that are not directly supported by the system, etc.

For development settings, direct observation could include sitting in on design review meetings. The purpose might be to elicit developer processes (e.g., testing routines, repetitive programming) that could be supported with reuse at the process level.

> *Example.* The Macintosh System 7 finder allows a user to either show the contents of a folder in tree form beneath the folder name, or to spawn a new window, with only the contents of the selected folder displayed. In an interview, a user said that he 'normally' uses outline mode, and 'rarely' uses the separate window, when navigating his file system. Direct observation of the work practice showed that in fact, he uses the separate window quite frequently. This information could be very useful in deciding which *feature sets* to include in an asset base later on.

➤ Instrument environments for data collection

An alternative or adjunct to the observation strategy above is to use automated support to derive instrumented data that includes process information about work practice in the domain. This has the advantage of being less intrusive than a human interviewer in the setting. On the other hand, data derived in this fashion must be interpreted with care because the contextual information obtained in face-to-face interactions is missing.

> Example (cont). The observation of the user of the Macintosh System 7 Finder might have been an aberration, and not really indicative of usage patterns in general. A more systematic and less intrusive way of collecting data would be to automatically monitor the finder, to record how often the tree form is used in comparison to the separate window. Such data could reveal work patterns such as 'use a separate window for data files, and a tree for application files', but would not be able to detect patterns like 'use the tree form when the room is crowded, and someone might be looking over my shoulder, otherwise use the separate win-

dow.'

➤ <u>Participate directly</u>

Another strategy to consider is to participate directly in domain work processes. This includes participation in development or use of domain functions. In either case, participant observation skills are needed to derive data simultaneously as the work is performed.

> *Example.* In performing the Outliner domain modeling prototype, the modeler got access to several commercial outlining programs and used them extensively to record the project data. Use of several different programs highlighted common and variant features in the implementation, terminology, and assumed usage settings for the various programs. It would have been very difficult to derive this data originally from either artifact analysis or "expert" interviews, although once certain insights were obtained, it was relatively easy to validate them with supporting data.

➤ <u>Perform experiments</u>

One effect of studying multiple systems for a domain engineering project is that some features that are distinguished in one system are not distinguished in another. These features will not appear in system documentation (since the distinction is not known), nor will human informants be able to provide information about them, since the distinction is not familiar in their work practice. An effective way to acquire data about such features is to perform an experiment, either on a running system, or as a thought experiment for a human informant.

> *Example.* In the Outliner domain, we have found that WebArranger has an entity called a 'folder', which can contain any number of entities called 'topics'. In the Mac System 7 finder, folders can contain other folders. Is this possible in WebArranger as well? The documentation does not mention this (it is unusual to document what *cannot* be done), nor have any of the users considered such a thing. A simple way to elicit the data is to perform an experiment; try to put a folder inside another.

When to Stop

You can stop *Elicit Data* when there are no more unfinished goals in the DATA ACQUISITION PLAN. This means that all the systems in the REPRESENTATIVE SYSTEMS SELECTION have been examined to a similar level of detail, in preparation for comparison during descriptive modeling.

Guidelines

- <u>Convert data to facilitate comparison and modeling only</u>. Creating new system workproducts is *not* part of the core ODM life cycle. For example, if it is determined that system designs will all be translated into $IDEF_0$ diagrams, the motivation should be support of comparative analysis across the artifacts, not an organizational decision that $IDEF_0$ should be the standard design language. The latter motivation can be considered in the DATA ACQUISITION PLAN, but is a separate concern from domain engineering.

  Excessive data conversion is a significant point of risk, for several reasons:

  — If artifact analysis is diverted into system modeling for its own sake, it can siphon away resources from the project to satisfy organization needs that are at best tangential to domain engineering goals.

— If project members are ill at ease with domain modeling concepts and tasks, there will be a natural tendency to slip back into familiar design activities.

- <u>Look for unanticipated information sources</u>. Each domain and each system will have unique information sources that are only discovered once data acquisition begins. These could include concepts documents, meeting minutes, informal checklists, expert system users who know all the undocumented workarounds and system bugs, authors of popular customizations, etc. do not let a pre-existing plan result in these serendipitous information sources being ignored. On the other hand, there is also a danger of following the trail of information sources and never coming to closure.

- <u>Consider data from previous domain engineering efforts</u>. Unless previous domain engineering efforts were performed using ODM or a comparable formal process, system artifacts produced will typically not be linked systematically to a deriving set of representatives. Thus, previous data cannot be allocated definitively to one particular representative system. Keep in mind that these efforts might not have gone under the name "domain engineering", but survey articles, consumer reports, etc. These will pose a particular challenge in the next task, *Integrate Data*.

- <u>Watch for diminishing returns</u> when looking at analogous artifacts across representative systems. As understanding of the domain increases and an initial model is built, data acquisition becomes increasingly an activity of scanning for new information or exceptions to previous results. Continuing past this point may be a waste of resources. But short-circuiting the data acquisition process may result in leaping to conclusions about assumed commonality across the domain, formed from initial data but not validated sufficiently.

- <u>Adjust for bias in data elicitation</u>. Many of the pitfalls and risks in data elicitation are apparent even in such simple tasks as posing a direct question to a human or looking up information in a document:

  — *Pose questions in understandable terms*. If you use terms unfamiliar to an informant, the quality of the answer will be unpredictable. do not assume an informant will be honest about what he doesn't understand. For an inanimate information source, this means knowing the terms to look up in a table of contents, index, or other guide to the text.

  — *Pose questions that the source can reasonably be expected to answer*. In addition to the obvious mistakes like looking for technical details in marketing materials, it is important to avoid forcing sources to respond in your terms rather than its own. For example, asking for statistics from someone who does not understand the notion of a 'representative sample' is likely to result in unreliable data. Avoid introducing new concepts unnecessarily; pose questions in terms with which the source itself has shown competence.

  — *Do not ask leading questions*. This is as important for documents as it is for people; in most complex documents, you can find some passage to corroborate almost any viewpoint. Resist the temptation to look for look for corroboration for answers you think you already know.

  — *Pay attention to the setting of a question or answer*. The same question can elicit different answers, depending on the setting of the information source. For example, a common mistake is to is to assume that developers have access to concepts of system use. Someone working in a real usage setting might well have a very different idea of system functionality than its developer.

- <u>Treat knowledgeable team members as informants</u>. If the domain engineering team includes domain experts it is important to document them explicitly as informants in the information

sources listed in the DOMAIN DOSSIER. The temptation will be to treat team members' domain knowledge informally. This will undermine the value of the DATA ACQUISITION PLAN as an accounting of the empirical basis for the final DOMAIN MODEL produced in the *Model Domain* phase, since informal and unverified domain knowledge will be incorporated into the model.

Having team members who can serve as real informants is a valuable resource in many ways. Other team members can practice informant interviewing with fellow team members in a low-risk setting. Interviewing teams can be strategically configured to include some members with domain expertise, to establish credibility with informants and allow for rapid information transfer. But utilizing such resources effectively requires an accurate and impartial assessment of the team members' status as informants.

- Follow the plan. One the danger during any elicitation session is that the investigator can be distracted by material that lies outside the DOMAIN DEFINITION. When interviewing informants, keep in mind that domain boundaries may not be evident or even intuitively obvious to them. Informants probably do not cluster their experience along these lines. Facilitate and guide the interview format to keep focused on data relevant to the domain. Otherwise a lot of extraneous data may need to be filtered, more painfully, in *Integrate Data.*

- Treat informants as potential customers. The ultimate goal of domain engineering is to develop an asset base that will get used. All contacts with informants, though ostensibly focused on information gathering, also serve to convey information about the domain engineering project, modelers' competencies and knowledge of the domain, and the nature of domain engineering itself.

  Paradoxically the best way of "selling" the project at this stage is to treat the informant as a valued source of information, rather than performing an oblique selling job on the asset base that will be built. People who feel they've had input into the content of domain models are more likely to feel a sense of ownership when asked to use assets developed on the basis of those models.

- Respect confidentiality. As with any data collection in organizations, some domain information will be sensitive. In the domain engineering context, data is being collected specifically for comparative purposes. This may raise issues such as risks of the data being used for performance evaluation or inter-divisional competition.

  ODM encourages maintenance of traceability information where feasible. Being able to trace data back to specific originating settings is important in developing the interpretive domain model. Unfortunately, this also makes anonymity difficult to guarantee. This is only one aspect of a larger set of complex ethical issues. At a minimum, any commitments made to informants must be respected by the modeling team, if the project is to maintain credibility within the organization.

- Pay attention to resistance. In domain engineering, practitioners are asked to share their expertise. Reluctance on the part of informants to provide some information may be due to a number of factors. For example, disclosure of expert knowledge may be seen as a threat to job security. Be alert for resistance, and respect informants' concerns and their rights to their knowledge.

- Leave some stones unturned. In each individual interview, and in the elicitation process as a whole, it is critical to be attentive for the perceived point of diminishing returns. If the modelers sense the interview has reached this point the informant may have concluded the same thing some time earlier. Rather than risk exhausting informants' good will or making them feel their time is wasted, it may be preferable to intentionally leave gaps that can be filled in

during later sessions, when DESCRIPTIVE MODELS will be presented for validation and interpretation by informants.

6.1.3 Integrate Data

In *Plan Data Acquisition*, we determined who would collect what data, where and how. In *Elicit Data*, the data was collected. In *Integrate Data*, data is recorded in a way that allows us to iterate back to planning, to determine whether more data needs to be elicited. The primary challenge in *Integrate Data* is data organization. For domain engineering, this challenge aims more specifically at data organization in support of the upcoming descriptive modeling sub-phase, where system commonality and variability will be modeled.
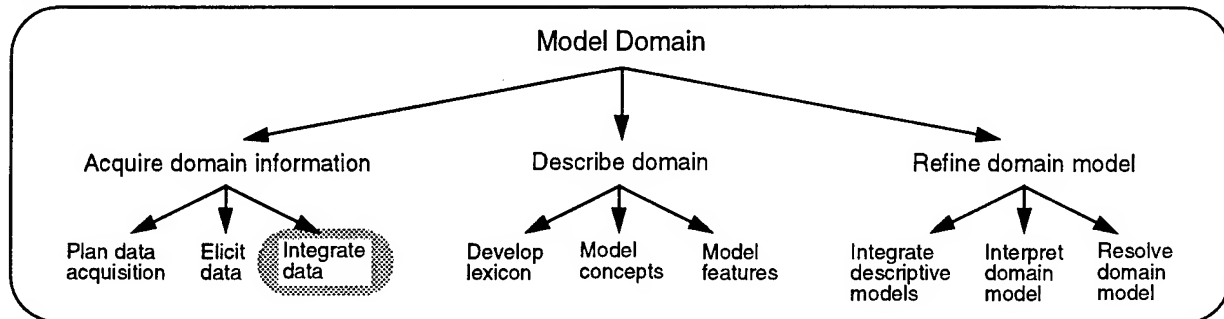
Exhibit 59. Integrate Data Process Tree

Approach

Many of the data organization activities can be achieved either by well-known data management methods (e.g., relational databases or spreadsheets), techniques from the Concept Modeling supporting method area, or data acquisition-specific data management methods taken from the Information Acquisition supporting method area. Any of these various methods, when used to integrate data, will consist of some way to find a 'bin' where a particular datum belongs (or to create it, if no appropriate bin already exists), and put the datum there. As these bins develop, there will be a tendency to move from general features of single systems to features shared by several systems. This is a natural development of data acquisition for domain modeling; the features that arise in this fashion will be passed on to the descriptive modeling sub-phase through the FEATURES OF INTEREST.

Workproducts

■ DOMAIN DOSSIER

The evolving "map" of the data sources from which the DESCRIPTIVE MODELS will be derived. The bulk of the dossier will be reports from knowledge elicitation sessions, along with an index to these reports, to facilitate access to them in the *Plan Data Acquisition* task and later during the *Develop descriptive models* sub-phase.

Also included in the DOMAIN DOSSIER is information about the historical or "genealogical" relations between representative systems. Some of this information was elicited as a basis for making the REPRESENTATIVE SYSTEMS SELECTION in the *Plan Data Acquisition* task. Additional information about system genealogy will emerge from the *Integrate Data* task as well.

■ FEATURES OF INTEREST

Short, informal sentences describing features of a system or many systems to which attention was drawn during data elicitation. Each short description is annotated with its source(s) and the setting to which it applies.

■ DOMAIN TERMS

Terms that have specific meanings for practitioners in the system settings studied, as deduced from the artifacts and informant data. The domain terms are linked to their occurrence in particular elicitation sessions. Syntactic relations such as *synonyms* are noted only where they are clearly suggested by the data in the domain or directly suggested by informants. Terms are not standardized in this task; this is done in the *Develop Lexicon* task within the subsequent *Describe Domain* sub-phase.

When to Start

Integrate Data can begin as soon as any DOMAIN DATA is available. Even writing down the answer to the first question asked of an informant can count as data integration. A thousand mile journey begins with a single step.

Inputs

- DOMAIN DATA. The record of the elicitation session that will be integrated into the dossier.

Controls

- DOMAIN DEFINITION. Provides criteria for determining which features may be of interest, and for determining how the dossier needs to be organized.

Activities

Integrate Data consists of organizing data; whenever you are deciding what to do with a particular datum or set of data, you are integrating data. In this task, observations about the systems are captured and placed in relation to one another, before starting to structure the commonalities and differences.

➤ Write up elicitation sessions

For each elicitation session (e.g., an interview with or observation of an informant, review of a user manual for an exemplar system) write up the results of the session. This involves recording the information source and whatever documentation of the elicitation session itself is required according to the DATA ACQUISITION PLAN; this information is stored in the DOMAIN DOSSIER. Besides this housekeeping, however, capture the real learning and insights about the domain that resulted from the elicitation session.

An experienced knowledge engineer has said that the hardest part of data elicitation is the write-up; this is where data acquired in an interview is brought together into a consistent report that can be read by other members of the domain engineering team, who were not present during the session. The same story holds for artifact analysis: anyone can read the same document over again; the trick of the write-up is to organize the information in a form accessible to domain engineers

198

who have not read the document. Details of how to complete a write-up depend on the particular session style and goals, but these general comments apply:

- *Keep the audience in mind.* The write-up should make the data accessible to people who were not part of the elicitation effort; know who will be reading the write-up, and make sure that the terminology or form in which the write-up is done can be understood by that audience.

  Example. An analysis session with the WebArranger software had the goal of reporting WebArranger native terminology for several outliner features. Since the intended audience does not know the WebArranger terminology, the write-up includes an explicit convention for indicating WebArranger terminology. Since the convention is not standardized throughout the domain engineering project, the convention is explained in the write-up itself.

- *Organize data according to session goals.* Information from an information source (informant or artifact) is normally organized according to some goal that is different from the goal of the domain investigator who is eliciting data. Domain engineers who will be reading the write-up will (hopefully!) share the goals of the investigator who elicited the data. Use these goals to organize the write-up.

  Example. A particular experimental session with the Macintosh System 7 finder wanted to determine how the hide/reveal functionality interacted with a distributed file system. If these objectives are not stated clearly in the write-up, confusion about the scope of the observations (do they also apply to non-distributed file systems? What qualifies as a distributed file system?) can result. In the worst case, this will result in repetition of the experiment by other domain engineers.

- *Use standard write-up formats whenever possible.* A picture can be worth a thousand words (as long as you do not have to spend as many words to describe the picture). Diagrams, figures, formatted lists, etc. can simplify the job of writing up a session, as well as the next round of *Plan Data Acquisition* activity, where the write-up is examined to find missing data that remains to be elicited. On the other hand, don't discount informal narrative prose if that is the most effective way of capturing the ideas. This text becomes part of the corpus of material that will provide input to the formal modeling steps downstream.

  Example. Reading the on-line help for the emacs outline mode, we elicit the facts that there is a function for making the next text body invisible, making the next text body visible, and making the next headings visible. We can write this information up concisely in a comparison diagram, as shown in Exhibit 60. The form of this diagram makes it clear that we are missing information about making the next headings invisible.

  This information can be used in the planning phase to set goals for further questioning, looking up information in documentation, or experimentation, to fill in the unknown information. The same information will be direct input to the modeling process. [2]

➤ Update domain terms

In addition to any information structures required by a selected Information Acquisition supporting method, any domain-specific terminology uncovered during data elicitation is recorded in the DOMAIN TERMS, along with a short, informal definition. At this point, DOMAIN TERMS should reflect observation of terminology found in the relevant work practice settings.

---

[2] In fact, the form of the matrix in Exhibit 60 is a good example of a simple kind of feature model representation, applied here to operations from one representative system; later, in the *Describe Domain* sub-phase, similar models can be created that integrate variants of operations observed across systems.

	Make invisible	Make visible
Next body	X	X
Next headings	??	X

Exhibit 60. Example diagram for emacs outliner features

Record the information source for the term if it is a new term. If the term is not new, consider whether the information source from which you have just gleaned the term is likely to be using the term in the same way. If you are not sure, err on the side of including the term again, with a different information source and your best guess at the meaning. Terms are not standardized in this task; this is done in the *Develop Lexicon* task of the subsequent *Describe Domain* sub-phase.

A helpful rule of thumb here is to distinguish *coarse* and *fine* distinctions in meaning. When the same term has subtle gradations of meaning, you are probably observing some of the normal terminological "churn" within a community of practice. If the same term has markedly different meanings, you may be observing a juncture between distinct communities of practice, or the signs of an active analogy domain.

> *Example.* In the above example, we found the term "next headings" in the emacs on-line manual. We have seen the term "next heading" in a PC-based outliner. Do these mean the same thing? Does "next" refer to a sibling heading at the same level, or the next heading at any level traveling downward in the document? And can the meaning of "forward" implied in "next" shift from downward to upward depending on the context of other operations? These are all possible shades of meaning that are clearly within the Outliner domain itself.

> As a contrasting example, as part of our initial search for outliner information sources, we have used a standard corporate library computer database to find articles about "Outlining" in past trade journals. We find a number of articles that refer to "Outlining" as a special typeface manipulation style, and as a graphic effect in certain high-end CAD/CAM applications. In each of these cases, we have clearly crossed a domain boundary. Yet the crossing point is worth noting, since there is clearly a deeper notion of "outlining as preserving the outer shape of a structure and hiding the inner detail; i.e., outline as silhouette" that is operative in these related domains and our own domain of focus. We could go back and add a suitable analogy domain entry in the DOMAIN CLASSIFICATION component of the DOMAIN DEFINITION.

➤ Condense FEATURES OF INTEREST

While integrating the information from an elicitation session, keep an eye out for features that appear to be of particular interest to some domain practitioners. Describe these features with short, informal sentences as entries in the FEATURES OF INTEREST. Note their source (e.g., a member of the modeling team inspecting a design, a comment noted during a walk through with a designer, a comment drawn from a design rationale document, a point made by a practitioner during an interview) and the setting in which they occurred.

Features can be brought to the investigator's attention either from direct mention by a human informant, or being highlighted in some document. As information from more and more exemplars is integrated into the dossier, some features will be found to occur in several systems. Although comparative modeling is not, strictly speaking, part of the *Acquire Domain Information* sub-phase, noticing common features is a natural result of integrating domain data. The hand-off to the *Describe Domain* is made when systematic comparisons are noted and recorded; in *Integrate Data*, commonalities are noted but not pursued.

> *Example.* While studying the Outliner domain, one of our informants tells us how the Macintosh System 7 Finder uses an icon in the left margin to indicate the disposition of a structure, whether it is full or empty, being displayed or hidden. We write this up in a short sentence (like the first sentence of this example), attribute it to the source who gave it to us, and enter it in the FEATURES OF INTEREST. While integrating data about other systems, we find that many of them share a similar feature. We annotate the feature with this information, but we do not launch an extensive comparative analysis.

➤ Record data in the dossier

The write-up of the data needs to be stored in a way that will allow it to be accessed by other members of the domain engineering team. Keep links to all materials generated from interviews or analysis, along with source materials (design documents, user's guides, etc.) in the DOMAIN DOSSIER.

Dossier entries can be organized in any way that is useful to the domain engineering team; the possibilities include:

- *Index by keywords.* One of the easiest indexing mechanisms is to use keywords; each dossier entry that mentions a particular word from the DOMAIN TERMS is linked to that word. A hypertext system can provide automated support for this sort of index.

- *Index by feature of interest.* Since some of the FEATURES OF INTEREST will be found to hold for more than one system, and any feature of interest might very well be specified as the goal of a particular data elicitation session, the FEATURES OF INTEREST are an excellent source of index material for the DOMAIN DOSSIER.

  Example (cont). Suppose that a complete experiment session had been organized around determining the various possibilities of the structure indication icon in the Macintosh System 7 finder. The report from this experiment could be found under an index for this feature of interest.

- *Index by information type.* If you have chosen an Information Acquisition Technique that provides different types of information (e.g., object diagrams, task diagrams, scenarios, time lines), then the information type can be used as an index.

  Example. An interview with a copy editor might produce a task analysis of the use of the Microsoft Word™ outliner. This dossier entry could be found under the keyword, 'copy editing', and the information type, 'task analysis'.

➤ Incorporate legacy models

A particular challenge arises when integrating data for domain modeling, in that results from other domain modeling efforts are often available. If a domain is sufficiently interesting that it has been selected for data acquisition and domain model, it is often the case that some modeling has been done before. Results of previous domain modeling efforts can appear in a number of forms,

including survey articles, comparative studies, standard lexicons, or just the private theories of particular informants. We term such data *legacy models* when codified in a form specifically referencing general concepts or multiple system comparisons.

> *Example.* In reading some trade articles about outliners, we find a good survey article that provides a thumbnail history of the evolution of outliner applications, written by a developer of a popular PC-based application. The article includes some prominent "families" of outline applications, distinguishing strict outliners from text-editor outline "modes" that will allow all kinds of anomalous outline structures to be edited. We note this as a source for a mini-taxonomy that comes from a practitioner in the field, rather than our own analysis.

Such data poses a particular challenge to the present domain engineering effort; they come from well-informed practitioners (and hence should not be ignored), but they also move well beyond the goals of data acquisition, in that they have already modeled commonality and/or variability in the domain. Special care must be taken when integrating this information into the DOMAIN DOS-SIER. How can we give such models the credit they are due, without simply copying them verbatim into the domain model?

There is no magic formula for determining how to incorporate information from legacy models into the current dossier. However, a key insight can help manage the relationship between such models and the current domain modeling project: every domain model (whether or not it is created via a formal process such as ODM) is a product of a particular set of stakeholder interests and biases. This means that legacy models and the descriptive models created as part of the domain engineering life cycle are models from distinct communities. A major advantage for the models derived in the ODM process is that the stakeholder context for the current domain selection has been documented in the *Plan Domain* phase. To understand what part of the legacy model is relevant to the current effort, you will need to recover some of the similar contextual background for the model. This is more difficult than for a system artifact easily associated with a single exemplar system. If developed via an informal process, the data from which the model was developed and its intended scope of applicability may be only implicitly documented. Consider the following points in assessing the legacy model as a source of domain information:

- *Does it model both commonality and variability?* Previous domain engineering models might not be attempts to model *variability* across systems in the domain, but may be "commonality only" system models. These models correspond to what common usage would term "generic" models. Such models cannot be expected to replace the commonality and variability based models produced by ODM.

- *Does the domain have the same boundaries?* Domains are often given short, informal names that conceal the subtleties of domain definition that were handled in the *Define Domain* subphase of the *Plan Domain* phase. The legacy domain is probably not intended to cover the same range of variability as the current effort.

- *What were the biases of the investigators?* The investigators who produced the information for the legacy model were subject to the same constraints of bias as described in *Plan Data Acquisition*. Try to determine how these biases were handled in the legacy model, and how this affects its relevance to the current domain model.

- *What were the stakeholder interests motivating the model?* The legacy model was created according to some stakeholder interests; since it is usually not possible to determine what these were, it is necessary to evaluate any information in the model in light of the current stakeholder interests.

➤ Identify analogous artifacts for comparison

Part of the data acquisition goals determined in *Plan Data Acquisition* were to determine whether similar features of different systems have been studied to a similar level of detail. In order to satisfy this goal, it is necessary to identify ***analogues*** between systems. Such analogues will be detected when the data acquisition team attempts to integrate information about several systems into the DOMAIN DOSSIER.

This activity marks a key transition point and hand-off between data acquisition and descriptive modeling. Identifying analogous artifacts can help clarify exit criteria for data acquisition (i.e., when analogous artifacts are described to a consistent level of detail). But some initial description is needed in order for modelers to perceive the analogous functionality.

Establishing a structural framework for comparison across representative systems, and integrating the structural and taxonomic frameworks are key challenges in this activity. For systems with closely aligned structure (e.g., members of a systems family) this existing structure may be used as a basis for correlation. Where structures are diverse, emerging FEATURES OF INTEREST will have to provide the structure for comparison.

When to Stop

You can stop *Integrate Data* when:

* There is no more DOMAIN DATA to be integrated. This means no further investigation was deemed necessary in *Plan Data Acquisition*.

* Analogous data appears to have been analyzed to a comparable level of detail.

* Domain terms and prominent features have been noted for the data collected.

Guidelines

* DOMAIN DOSSIER structure. The organization of the DOMAIN DOSSIER produced in *Integrate Data* serves two purposes; first, it is used in *Plan Data Acquisition* to determine what further data needs to be elicited. Ultimately, however, it will be used to provide access to the dossier materials for the members of the modeling team who will be developing the descriptive models later on. When modifying the structure of the dossier, think about how it will benefit one or both of these goals. In general, both goals can be served by structures that facilitate comparison, either within a system or between systems.

    Depending on the size and complexity of the project, the dossier might require considerable structuring. One option to consider is to use taxonomy-based techniques from the Concept Modeling supporting method area to model the information sources and their relationships within the dossier. For example, one could define a taxonomy of different kinds of informants, different kinds of workproducts studied as artifacts (requirements documents, designs, etc.) This is an area for further research.

* Let the DOMAIN DOSSIER stand on its own. The information in the dossier, along with its structure, should have everything in it to be understood by a modeler. It can often be the case that the domain engineers doing the modeling are not the same people who have done the data acquisition; do not make them repeat the elicitation activities that were done here.

    This is a useful goal to keep in mind in creating the DOMAIN DOSSIER. That is, document the data as if others will be the ones to codify it in more formal models. This will encourage you

to capture as much as possible informally. Of course, this goal is not really attainable. There will still be a great deal of learning about the domain that results from the experience of immersion over time in the data, learning that cannot be transferred via documentation. Therefore, it is important if at all possible to have some continuity between the people doing data gathering and modeling activities.

6.2 Describe Domain

In the previous sub-phase, *Acquire Domain Information*, a rich set of data has been gathered for a set of representative systems in the domain. Some interpretation and filtering of this data has also been performed; but the emphasis has been on gathering data about individual systems, preparing it for comparative analysis, and validating the data and initial interpretations with informants.

Model Domain

Acquire domain information   Describe domain   Refine domain model

Plan data acquisition   Elicit data   Integrate data   Develop lexicon   Model concepts   Model features   Integrate descriptive models   Interpret domain model   Resolve domain model
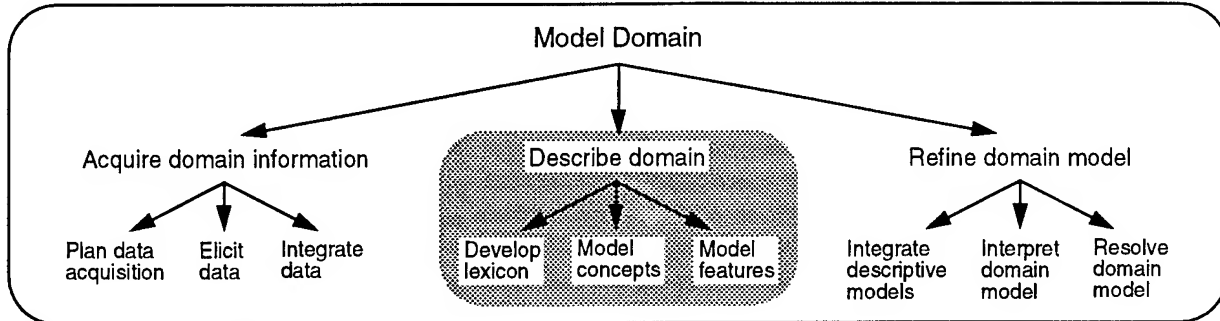
Exhibit 61. Describe Domain Process Tree

The primary purpose of the *Describe Domain* sub-phase is to distill this domain information into formal models that represent the **commonality** and **variability** across the representative systems studied. This picture of the differences between systems provides an essential intermediary form between the direct domain information (including system artifacts) and the eventual reengineered assets that may be created in the *Engineer Asset Base* phase.

In this sub-phase we begin to derive the full benefit of studying multiple systems in parallel. It sets the groundwork for uncovering rationale and implicit context in the system artifacts, a focus of the subsequent *Refine Domain Model* sub-phase. Understanding this rationale is essential to the goal of reengineering artifacts to be reusable in a broader set of applications.

The transition from data acquisition to modeling involves a shift from gathering the "raw material" about individual exemplars to describing semantic relationships *between* elements in the exemplar systems. There is a corresponding shift from relatively informal to more formal analysis and modeling activities. Modelers will typically develop a number of separate models, selected and tailored to suit the PROJECT OBJECTIVES and the particular characteristics of the domain of focus. We intentionally defer interpretation and introduction of innovative ideas for the domain. These are the purview of the following sub-phase, *Refine Domain Model*.

Approach

Domain modeling can be thought of as defining a formal language for describing domain entities and behavior. Based on the analogy to a language, lexicon terms provides the *vocabulary* for this language, concepts provide the *semantics*, and features correspond to actual *sentences* or *statements* in the language. Unlike a natural language, however, the domain language produced in domain modeling creates a fixed repertoire of statements that can be made about domain entities.

This sub-phase focuses on *descriptive modeling*, which is both a process observing certain formal constraints, and a particular approach or mindset somewhat different than the traditional problem-solving approach in engineering. Formally, terms, concepts, and features described in the DESCRIPTIVE MODELS are justified by the *precedent* of occurring in at least one system or system requirement included in the REPRESENTATIVE SYSTEMS SELECTION. The *feature profile* of an artifact is the set of feature statements that hold for the artifact in question. There is no implicit commitment to provide assets to support any particular setting studied in this sub-phase. In ODM,

these formal guidelines are used in part as techniques to help reinforce the attitudinal shift to descriptive modeling.

In the following paragraphs we will provide some discussion of how this type of modeling differs from more familiar system modeling approaches. In particular, we discuss the key distinction between taxonomic versus system modeling, and between concepts and features.

Taxonomic versus System Modeling

In designing or implementing a program, we might model a given operation in terms of the arguments or parameters (inputs, outputs, saved state, etc.) This would yield a model of a single operation. If operations interact, exchange data, etc. we could build a more thorough functional or structural model of the program to represent this. This all falls under the category of a *system model* from the ODM standpoint. You may need to have one or more of these models available as *artifacts* to work from; but such models are not in and of themselves *domain models* in the ODM sense.

If we only model the functionality of a given operation, we are doing software engineering, not domain engineering. This is true regardless of the quality of our design or implementation. On the other hand, if we only compare applications in broad, high-level terms that do not discuss specific features and combinations of features, we are doing a comparison that is little more than a kind of product review. It is not an adequate basis for reengineering components to be used across multiple applications. To accomplish this latter goal, we must have a detailed model of the **commonalities** and **variabilities** across the applications.

This means that, typically, we will need to model both specific operations and applications that implement these operations. Each application can be viewed as supporting or implementing a *set* of operations. Different applications support different sets of operations. The set of operations supported by a given application is usually not an arbitrary conglomeration of operations. There are constraints and inter-dependencies between the operations.

> *Example.* In the Outliner domain, we have examined two different representative systems. One is an outliner-oriented directory browser, another a text editor application. Both of these systems provide functions for "expanding" and "collapsing" outline headings. Both provide one version of this operation that operates on the immediate heading level only, and another variant that operates on the full sub-tree below the current heading. However, the text editor provides an operation to incrementally expand lower levels of the tree while staying positioned at a given heading level. The directory browser does not support this variant of expand and collapse.
>
> Suppose we develop a model where various "flavors" of expand and collapse operations are represented in *semantic relation* to each other. That is, the concept "expand" occurs in a taxonomy, representing a class or set of operations that all satisfy the semantics of expanding some part of an outline structure. "Expand all" might appear as a *specialization* of "expand" because it is a more specific operation of this kind.
>
> A more detailed taxonomy of this kind, elaborated to cover all variations observed across a given set of exemplars, would constitute a *concept model* focused in the area of "outliner operations."[3] The taxonomy might include some flavors of the operations that are mutually exclusive; i.e., when an application implements one flavor it almost never implements the other flavor. In this way the model represents both commonalities and variabilities in domain functionality across multiple systems.
>
> We need to go one step further, however. Our model will help us make comparisons between

Domain
definition

Project
constraints

Project
objectives

Domain
terms

Develop
lexicon

Domain
lexicon

Domain
lexicon

2.2.1

Concepts
of interest

Model
concepts

2.2.2

Concept
models

Domain
dossier

Feature
models

Descriptive
models

Model
features

2.2.3

Features
of interest

New domain
terms

Exhibit 62. Describe Domain IDEF$_0$ Diagram
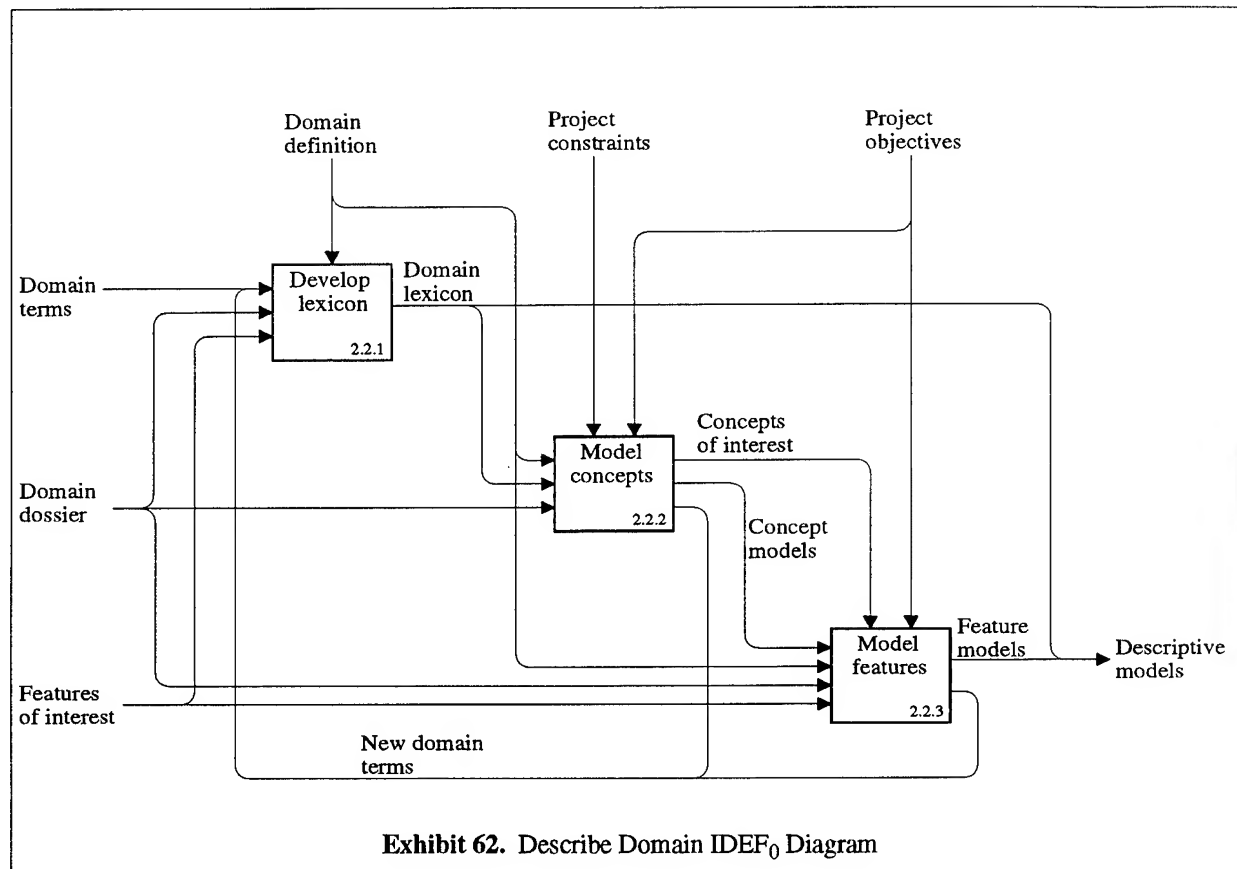
different operations within the domain. But we need language that will help us differentiate one outliner *application* from another. Thus, "Outline operation" and "Outliner application" will be two important, if fairly broad, CONCEPTS OF INTEREST for the Outliner domain.

Software system models generally do not need to be able to represent both operations and applications that implement operations. In a sense, the entire model describes the application, which is the "invisible frame" or context of the model. This is also more complex distinction than that which holds between two individual operations. This makes the requirements for software domain modeling more complex than either software engineering on its own or simple taxonomic modeling of different kinds of things in general.

Concepts

A *concept* is an abstract idea generalized from particular *instances*. Concepts can only be talked about through shared definitions. We cannot "get at" concepts directly; only through social interactions and agreements, mediated by representations.

People define concepts by grouping a set of individual descriptions of phenomena (exemplars) according to some principles for determining whether or not an individual description is an exem-

---

[3.] At this point, an object-oriented enthusiast might object that we are talking about "nothing more than" an object-oriented class library. There is a similarity, to be sure. A good class hierarchy should reflect a clear conceptual model; but not all concept models would be appropriately implemented as class hierarchies. For example, most OO programs are structured as classes and methods; in our model here operations themselves form the hierarchy.

plar (defining rules). A concept definition *embraces* an individual description if the description is an exemplar for that concept.

- The set of all possible individuals embraced by the concept definition is the concept's *extent*. The set of individuals considered and determined to be embraced by the concept is the *exemplar set*. The exemplar set is always a finite subset of the extent.

- The minimal set of properties that provide a decision procedure for new individual descriptions to determine whether or not they are embraced by the concept is the *intent*. Just as we can only get at the extent through the exemplar set, the intension can only be approximated by an articulated set of defining rules.

A concept "lives" in a set of primary tensions: between the extent and the exemplar set; between the intent and the defining rules; and, most accessible and useful, the tension between the exemplar set and the defining rules (that is, the tension between the intension and extension). Working with this latter tension, in fact, is a primary way to incrementally refine both the exemplar set and the defining rules so that they better capture the intended concept to its full extent.

These terms should look familiar: they are directly analogous to the work done in the *Define Domain* sub-phase for the domain as a whole. In some senses, the domain could therefore be viewed as a single high-level concept. The *Model Concepts* task focuses on the constituent concepts that together define the semantics of the domain.

A concept is represented in a Concept Modeling supporting method (or *formalism*) by a "concept-capturing construct." For convenience, we will overload the term *concept* and use it in lieu of "concept-capturing construct." That is, we will speak of models that contain concepts as elements, but this terminology is merely short-hand. It is important to understand this underlying picture at the outset, because we are trying to capture these fuzzy and ill-defined concepts with representations. This terminology allows us to describe the process independent of an assumed formalism.

Features

The word *feature* is interpreted in widely different ways by different practitioners and methodologists. If we think about familiar, Consumer Reports-style product comparison charts, features are attributes by which a class of similar products are compared and evaluated. In other domain analysis approaches (e.g., FODA), a feature is typically something that might be termed an "End-User System Operation." Use of this term in the ODM context extends these more familiar and informal intuitions of the term.

We will provide a more formal description of this term as understood in the ODM context in the *Model Features* task section. For the purposes of this high-level discussion, a feature can be thought of as "a difference that makes a difference." This raises the questions: a difference *in what*? That makes a difference *to whom*? To fill in the missing terms: a feature is *a difference* observed by modelers among multiple exemplars of a *concept of interest*, *that makes a difference*, i.e., is of interest to some domain stakeholders. In particular, a feature should be relevant to some domain practitioner.[4]

Concepts versus Features

[4.] In the long term, features will become differentiators that help asset utilizers select specific assets. However, we are doing descriptive modeling at this point and have not yet decided who the customers for the asset base might be.

A given Concept Model supporting method used may impose its own distinction between the types of things modeled as concepts and as features respectively. The distinction between concepts and features as the terms are used in the core ODM process model is a fairly formal one that is focused on *how the process changes when you shift from concepts to features in modeling.* Essentially the distinction is between elements in the universe of discourse (classified by concepts) and properties of those elements (captured by features). Any **concept model** (e.g., a taxonomic model) can "sprout" features as differentiators of the concepts in the model. Any feature model can be extended to a model with explicit categories for the things differentiated. However the distinction is maintained, domain modelers need to preserve clear distinctions between concepts and features to manage the process smoothly.

As an example is provided below of the relationship between concepts and features, let us look at a simple example from the domain of Chairs. Suppose our exemplar set includes three chairs of three different colors. Our concept model for Chair *could* have been extended with specializations like Green Chair, Red Chair, Blue Chair. This could all have been done efficiently in one model as long as these are the only distinctions we want to make in the domain of chairs. However, as soon as we want to distinguish other aspects (e.g., Victorian versus Edwardian chairs) we will find that having "embedded" color attributes in the basic taxonomy for chairs has tied our hands. Green chairs cannot also be red chairs, but green chairs can be Victorian, etc. This creates the need for multiple categories, multiple inherited categories, etc. At this point, a better option might be to consider "green," "red" etc. as "features" of the Chair concept.

Note that there is no absolute criterion by which to say that the color attribute "should not" have been included in the basic concept model. It depends on the entire configuration of concept areas to be modeled.

In order to separate out this aspect of Chair, we need to name the "feature category" of which "green" and "red" are specific values. In this case, it's pretty easy to call this Color and to consider it an attribute. However, suppose we got more interested in the chemical composition of the paint used on the chairs. Then what we had called Color would turn out to be a first approximation of a feature better named Paint, and the relationship to the concept Chair becomes closer to a component (part-of) than an attribute or property relation.

One could then imagine modeling all kinds of attributes of Paint besides its color. In other words, Paint can, in the blink of an eye, become a concept in its own right and not just a feature of the Chair model. *The only thing that prevents this shift is our maintaining focus on our original concepts of interest as part of our modeling "discipline."* This focus, in turn, can be traced through all the ODM workproducts back to the strategic, stakeholder interests in doing the modeling in the first place. The connection between organizational and technical issues is thus more than a nicety: it turns out to be essential grounding to escape dilemmas that arise in the midst of the formal modeling process itself.

ODM vs. FODA Notion of Features

For readers familiar with the central use of the term "feature" in Feature-Oriented Domain Analysis (FODA), it may be helpful to understand the differences in usage in detail. In FODA, the term feature is typically used for an end-user operation for a system in a domain (e.g., the operation to expand an icon to an open window in a window manager application). Evolution of the method has extended the distinct types of features considered: e.g., operational features, non-functional features, development features, etc. From the ODM standpoint, the FODA repertoire of types of features could be used as a Concept Modeling supporting method, specifically as a **concept starter set**. The use of an AND-OR graph formalism would be an associated **concept modeling formalism**. Note also that while FODA associates the repertoire and the formalism, the types of features could be modeled in other ways, e.g., with semantic network-style taxonomies, and con-

versely, different concept areas could be modeled with AND-OR graphs.

Thus, while the terminology may differ because of different purposes and scopes of the method, there is no fundamental incompatibility between the methods. ODM provides a framework that allows for a wide degree of discretion in concept focus which includes the typical FODA areas of concern for feature modeling as a special case.

A possible point of confusion, however, should be noted: what a FODA modeler might call a feature could be called a concept in ODM! In ODM, end-user operations might also be a primary area of attention. However, ODM does not prescribe a certain set of concept areas to address as "features": if such operations were a specific focus for the modeling effort, they would typically be modeled as *concepts* under a broad concept area like "End User Operations." To clearly delineate this concept area, we would have to specify which aspect of the operations we intended to describe (e.g., the "mental model" of the system user as they interact with the application, the software module that implements the function, etc.) Clarifying the ontology of the concept area in this way is one of the activities of the *Model Concepts* task. Relevant features *for this concept area* would then emerge as part of the modeling process.

Results

- The DOMAIN LEXICON serves as the "gatekeeper" where all terminology specific to only certain exemplars is linked to terms used in the models.

- The CONCEPT MODELS document all the important "domain-native" concepts and describe their semantic relations in unambiguous ways. Additional concepts (e.g., higher-level classification support concepts) can be added by modelers but whatever new language thus introduced must be filtered through the DOMAIN LEXICON as well.

- The FEATURE MODELS extend information in the CONCEPT MODELS to provide differentiation of key concepts, so that we know not only that they are different but *how* they are different. Later, in the *Refine Domain Model* sub-phase, we will go on to document our best theories about *why* they are different; that is, why different exemplars exhibit patterns of commonality and variance with respect to these concepts.

Process

As shown in Exhibit 62, the *Describe Domain* sub-phase consists of three tasks:

- In the *Develop Lexicon* task, modelers document and manage domain terminology reflected in the artifacts and transcripts of informants' terminology. By normalizing the syntactic domain terms in the DOMAIN LEXICON, the descriptive models can be focused on essential semantic relationships. Lexicon development is considered part of modeling rather than data acquisition because a certain amount of comparative modeling is required to identify synonymous terms and select an appropriate standard term to use in the DESCRIPTIVE MODELS.

- In the *Model Concepts* task, lexicon terms are mapped into CONCEPT MODELS that define semantic relationships among domain terms. Separate models created in this task are validated for internal and cross-consistency, and for completeness and conciseness with respect to domain information.

- Concept models provide the groundwork for the *Model Features* task, in which models of domain *features* are developed, which differentiate domain systems and artifacts.

210

What makes the *Develop Lexicon* task different from the subsequent modeling steps? The lexicon is intended to strip away differences that do not make a difference to domain practitioners. In *Model Concepts* and *Model Features* we will capture the differences that do make a difference in terms of the semantics of the domain. The DOMAIN LEXICON serves as a filter for domain terminology from the DOMAIN DOSSIER into the DESCRIPTIVE MODELS. Developing the lexicon separately and as a precursor to concept and feature modeling means the lexicon will be more likely to catch important domain terminology that might not fit within the set of models being developed; this is a good completeness check for the modeling process as a whole. The judgment call required is as follows: If X and Y are terms in the lexicon, do we cluster them as synonyms within the lexicon, or do we keep them as separate terms and migrate them both into the concept models? While there is no mechanical answer to this question for all cases there is still value in having the two separate forms.

> *Example.* Most text-oriented outliners have a name for the entity that appears at a given level in the outline, and that can contain other entities. This entity has various names in various exemplar systems, however. MSWord might call it a "heading," other programs might call it a "header," another program might call it a "parent."

The *Develop Lexicon* task is also different from subsequent modeling steps because the lexicon is modeled as a flat structure. The lexicon is flat so that you can get at a term directly without knowing anything about its semantics.

This is an imperfect process. The method is designed to be robust, so that initial errors in the process get caught and adjusted later on. The initial pass through the lexicon is still a useful step, even if some decisions get retracted later.

The primary value of the concept/feature distinction is to help maintain focus during modeling activity. There is no a priori way of determining what should be modeled as a concept versus a feature. The distinction is almost purely operational, that is, a distinction of process and focus. Concepts are "stuff we are interested in." Features are "interesting things about the stuff we are interested in." If you allow the shift of focus to features as items of interest in their own right, they could well become concepts which have, in turn, their own features, and so on ad infinitum. This loss of focus, from a process point of view, can be traced precisely to the moment where this shift occurs. By defining what the CONCEPTS OF INTEREST are and modeling features as needed for these concepts only, we tether ourselves in place in the conceptual landscape where domain modeling is performed. The distinction between concepts and features thus also provides closure criteria for the *Describe Domain* sub-phase, by setting bounds around the portion of the model that must be "accounted for." Otherwise, it is difficult to apply formal validation or termination criteria to the model.[5]

Guidelines

The movement between concept and feature modeling can be very rapid. In fact, you could move between both tasks within the space of a single modeling session. It is possible to work either from concepts or features, and to determine this on a model-by-model basis or more globally for the overall process. Two broad styles of modeling can be identified in this sub-phase: concept-driven and feature-driven modeling.

[5.] The concept/feature distinction is also useful because there are different constraints on the kind of models that can be used for concepts and features respectively. Models for features, in particular, can be extremely small and simple.

In *concept-driven* modeling the initial emphasis is on modeling the semantics of lexicon terms. Once the concepts are established, then features are added to help motivate the distinctions between the various concepts. One advantage of this approach is that you can discover features that were missed (e.g., features that were not elicited as FEATURES OF INTEREST during the *Acquire Domain Information* sub-phase). (Note that at this stage, each identified feature must still be precedented within some representative system.) Also in this style, features can be defined that link relevant concept entities within or across specific CONCEPT MODELS. In this way features can serve as an initial integration mechanism for the CONCEPT MODELS. Working forward from the DOMAIN LEXICON to concepts, then on to features may yield finer-grained FEATURE MODELS, but may also get off target and fail to identify features of critical interest to domain informants.

In *feature-driven* modeling, candidate features from the DEFINING RULES and FEATURES OF INTEREST are converted into canonic lexical form using terms from the DOMAIN LEXICON and then mapped directly to features. The resulting features can then be analyzed into its constituent concepts. Working backwards from features (derived directly from FEATURES OF INTEREST) to concepts can result in more sparse and concise CONCEPT MODELS, which grow only those concepts necessary for defining features. One risk is that some important underlying concepts will not be identified. Also, detailed semantic relations between features will be more difficult to establish.

Iteration between concepts and features "views" on a per-model basis may also be a useful sequence. Regardless of the style and sequence chosen, it is probably a good idea to go through the integration process on at least a few partial models before going too far down the path with multiple modeling activities, to make sure the coordination process is sound. Use the DOMAIN LEXICON to control divergence of models, and frequent team reviews to make sure that modeling efforts are applying consistent conventions.

6.2.1 Develop Lexicon

In *Acquire Domain Information*, we have gathered and/or generated a lot of information about domain exemplars. We may have looked at system designs, studied requirements documents or user manuals, interviewed system operators, or read survey articles written by experts in the field. Observations, interviews, or field notes from our own experiments with trying out systems in the domain have been captured in appropriate reports. The DOMAIN DOSSIER has captured pointers to all this information. We have a lot of data, and ideally we know where each piece of data came from and our rationale for gathering it.
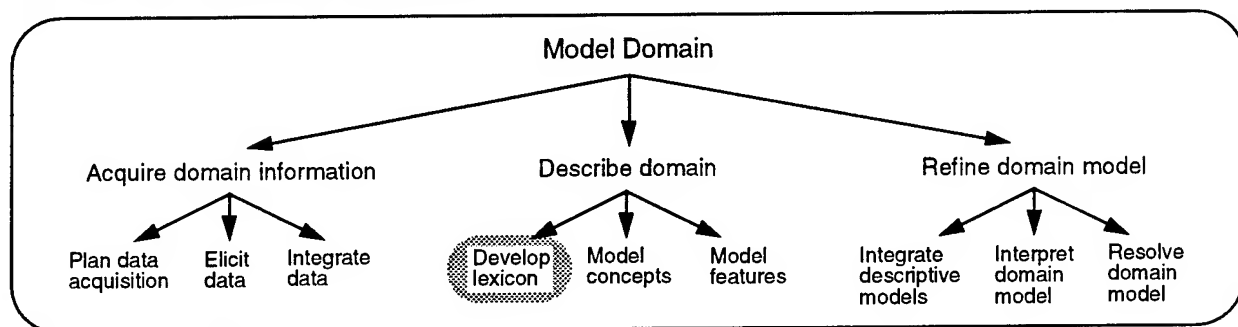
Exhibit 63. Develop Lexicon Process Tree

Now we want to model the commonality and variability of the data we have elicited. Before doing formal modeling, though, we want to screen out terms that reflect "uninteresting differences." This will make the models themselves easier to build and understand.

This step is often referred to as building a data dictionary, glossary of terms, controlled vocabulary, or lexicon. In ODM, we use the term *lexicon* because it best suggests the idea of collected terminology specific to a particular technical field or discipline, as opposed to, for example, terms used within the design of a single system.

Ideally, a lexicon ought to be directly owned by and useful to domain practitioners with no required knowledge of a special model formalism. The process of building a domain lexicon can itself be a significant intervention within stakeholder organizations because it may raise more fundamental issues in the course of comparing and normalizing terminology. A lexicon's usefulness can be validated by seeing if it helps foster conversations between people using different terms in different ways.

Building the lexicon is not trivial for most domains:

- There seems to be a natural tendency, in discussing a new subject and thinking about it from different perspectives, to constantly change the words one uses. The *Develop Lexicon* step signals a transition to a more disciplined use of a controlled vocabulary. Getting careful about our language is a first step in modeling.

- New terminology will usually be needed in modeling. There must be a way of explaining this new terminology, and the need for it, to domain practitioners. Also, since modelers may come from different communities of practice than domain practitioners, they may bring their own preferences for terminology and reflect these preferences in terms used in models. There is a difficult tradeoff involved in finding suitably neutral, yet familiar, terms.

Approach

The approach to lexicon development in ODM directly addresses the concerns above. A key element in the ODM approach is to separate the lexicon as a kind of syntactic analysis phase from the subsequent concept and feature modeling tasks which emphasize semantic analysis. This approach allows variant terms to be traced back to the setting from which they were derived. This makes the lexicon a useful bridge back to domain informants, who can interpret and validate the lexicon without knowing the semantics of the taxonomic modeling representation chosen for the CONCEPT MODELS and FEATURE MODELS. In addition, it enhances the lexicon's direct value as a kind of "Rosetta Stone" to facilitate communication among practitioners that may have previously had little work-related contact.

The basic approach is to identify terms within domain data, associate terms that play the same semantic role in different settings, and propose common terms to use in the models that can be mapped back to the corresponding variant terms. Terms can be drawn from all domain information sources, including development artifacts (code, design, tests), documentation, literature, and terminology noted in interviews and meetings with domain informants.

In the ODM perspective, modelers with their background and language are treated even-handedly as another community of practice within the overall project context. Their terms are not privileged with respect to domain-specific terms. By documenting which terms are introduced in the modeling process, modelers are actually left freer to invent terms as needed while preserving the ability to translate them back into terms familiar to the domain practitioners, or to change them easily.

The resulting DOMAIN LEXICON provides these benefits:

- A *description of key domain terms* independent of the taxonomic modeling representations used to develop the DESCRIPTIVE MODELS. This facilitates communication with informants

213

who may not be familiar with these taxonomic representations. Glossaries and data dictionaries are documents with which many technical workers are comfortable. The lexicon may therefore provide an early bridge to domain informants otherwise difficult to draw into the modeling process.

- A *mapping between different language communities* to provide practitioners in these communities (e.g., different developer or user environments) a basis for consistent communication. This terminology resolution may itself be a valuable outcome of the project. (Note that this does not mean imposing a standard set of terms, but providing a cross-translation between sets of terms as used.)

- A *controlled vocabulary filter* to the model-building process, so that syntactic variation does not complicate the semantics in the models.

Most domain analysis methods advocate gathering domain terminology in some consolidated form. One method that addresses lexicon development in some detail is DAPM [Prie91a], which borrows lexical analysis techniques from the field of library and information science and includes a detailed process model for lexicon creation. There are a variety of other lexical analysis techniques available from various disciplines that can serve as a Lexical Analysis supporting method for this task. Such supporting methods can provide the following:

- Suggested syntax for the domain language of which the lexicon forms the vocabulary;

- Techniques for extracting lexicon terms from a textual corpus (e.g., from requirements analysis or descriptive data about the domain);

- Techniques to address problems like term definitions that co-reference other lexicon terms;

- A starter set for lexical relations beyond the synonyms suggested by the core workproduct structure described above.

Workproducts

■ DOMAIN LEXICON

The DOMAIN LEXICON collates terms drawn from the domain, annotated to reflect information about each term with domain-specific meaning. The DOMAIN LEXICON indicates the domain of focus; it might also include links, if appropriate, to lexicons for related domains as described in the DOMAIN CLASSIFICATION or DOMAIN INTERACTIONS.

The lexicon consists of a set of entries that group terms with synonymous semantics with respect to the domain scope. The structure of each entry is as follows:

- One or more domain terms which are, by virtue of being clustered within a single entry, declared to be synonymous with respect to the domain.

- Each term in the lexicon has one or more *precedent* links to some material in the DOMAIN DOSSIER that shows the "term in use." For example, the link may point to a definition or a reference to the term in an article or its documented use by an informant during an interview.

  Note that this is not the same as having a link to an instance of the thing a term describes. For example, if there were a lexicon entry for the term "outline document" the precedent link might point to a user manual where this specific term is used. It need not be pointing to an actual outline document as an artifact studied and noted in the DOMAIN DOSSIER, and if there

were such an artifact there would still need to be precedent to the terminology itself.

- A designated *canonic term* for the set of terms clustered in the entry. This is a standard term to be used in normalized references in lexicon definitions for other terms in the *term cluster*. This standard term is also the term used in the DESCRIPTIVE MODELS. The canonic term could be one of the terms in the term cluster that is promoted to mayoral status; or it could be an adapted term devised by the modelers.

- There may be optional other lexical relationships that link the term to others in the lexicon. Examples of these other relationships include antonyms, oppositional pairs and inverse operations (e.g., "promote" versus "demote" headings in the Outliner domain). If a Lexical Analysis supporting method is used it may specify particular links of this sort.

When to Start

You can start adding domain terms to the DOMAIN LEXICON at almost any point from the start of the *Model Domain* phase. However, the specific role of the DOMAIN LEXICON as a filter for controlling vocabulary becomes operative once:

- The REPRESENTATIVE SYSTEMS SELECTION has been made. Each representative system may represent a distinct language community to be coordinated through use of the DOMAIN LEXICON.

Inputs

- DOMAIN DOSSIER. Used as the primary source for lexicon terms. The dossier contains references to a variety of information sources consulted during the *Acquire Domain Information* sub-phase. Domain terminology may have been introduced from any of these information sources: e.g., language used by an informant during an interview, terms defined in a survey article, or terms from a data dictionary or design document from a legacy system.

- FEATURES OF INTEREST. Another potential source of lexicon terms. In particular, the feature descriptions may be a source of compound terms (phrases that need to be handled as discrete terms).

- DOMAIN TERMS. The list of terms produced in the *Define Domain* sub-phase of the *Plan Domain* phase. In addition NEW DOMAIN TERMS introduced in the CONCEPT MODELS and FEATURE MODELS are fed back into the DOMAIN LEXICON via this input.

Controls

- DOMAIN DEFINITION. The primary filter on what terms go into the DOMAIN LEXICON. In general any source of terminology will include terms that are outside the intended scope of the domain. Continual reference to the defined domain boundaries is needed in order to ensure that each lexicon term is a term that has specific meaning within the domain setting.

Activities

The overall pattern of activity for this task is as follows:

- There is an initial planning step for the lexicon as a whole, where the intended audience, usage and validation strategy for the lexicon is determined.

- Then an initial version of the lexicon is developed, sufficient to provide a first pass that will allow further modeling to be done in a consistent way. This process is basically one of filtering, "winnowing" and extracting of terminology from a variety of information sources. Within this development phase, a "mini-life cycle" can be followed for each lexicon entry (synonyms clustered with their selected canonic term). A variety of specific sequences can be followed to complete these entries.

- The lexicon should then be maintained for the duration of the domain engineering life cycle, so that at all times it reflects a kind of flattened view of the terminology used in the DOMAIN MODEL.

The following paragraphs describe the activities for this task in more detail.

➤ Record modelers' informal domain vocabulary

Domain modelers knowledgeable about the domain may have a large repertoire of existing terms as part of their informal knowledge. If so, document these terms. It may be useful to prototype this "modeler as domain informant" lexicon prior to extensive examination of other information sources. Later it should be cross-referenced to tracked information sources.

In addition to providing a quick entry to the modeling process, the prototype lexicon can provide useful process documentation. For example, prototype terms eventually excluded from the lexicon, and lexicon terms that were not originally included in the prototype, show the level of domain knowledge within the team at the start of the modeling effort. Prototype terms should therefore be uniquely distinguished from later additions to the lexicon.

➤ Examine lexicon artifacts

Lexicon data, particularly definitions for terms, can often be derived directly from artifacts like data dictionaries created as part of various structured analysis and object-oriented methodologies, or published glossaries. Many of these sources of information should have been identified in the DOMAIN DOSSIER. These sources should be reviewed and culled for domain-relevant terms. Note that often the scope of these documents will be broader or narrower than the domain scope.

➤ Identify terms in domain information

Highlight and delimit terms within DOMAIN DOSSIER materials used in descriptions of domain functionality. For data elicited from informants, key terms should have been noted in the interview report. For artifacts, some analysis will generally be required. Working on parallel artifacts across representative systems will help make variant usage of terms more evident.

Terms can include phrases as well as single words.

> *Example.* In the Outliner domain, the term "heading text" is a compound term that has a distinct meaning. "Heading" also has a distinct meaning; whereas "text" might not be included in the lexicon as an individual term because it does not have domain-specific meaning as an isolated term.
>
> Note that the semantic connection between "heading" and "heading text" will not be captured in the DOMAIN LEXICON. The fact that "heading text" can be considered a component of a "heading" will be captured in the *Model Concepts* or later tasks.

216

➤ Document terms

For each term discovered, create an initial entry in the DOMAIN LEXICON. Note the source of the term and its precedent (i.e., examples of the how the term is used for each term in the lexicon.

Write brief definitions for the terms. For *prototype modeler* terms or terms gathered from a lexicon-like source, the definitions should be directly available. Otherwise, write definitions for the terms drawing on the insight gained from the *Acquire Domain Information* sub-phase, keeping in mind the target audience. Definitions in the lexicon should be written with a particular audience in mind and the audience should also be indicated in the workproduct. In particular, it is important to decide up front whether the definitions should be useful for modelers unfamiliar with the domain, for knowledgeable domain practitioners seeking to understand variant terms, or for novice practitioners to use in learning about the domain.

Highlight use of other lexicon terms within definitions. Such references to other terms in a term's definition may be helpful, but be cautious of building in circular definitions.

➤ Check for synonyms

Once the definition for a starting term has been written, search the DOMAIN DOSSIER and FEATURES OF INTEREST for *synonyms*, variant terms that appear to have the same domain-relevant meaning. Add these into the DOMAIN LEXICON as appropriate. This activity involves iteration from the lexicon back to information sources, using the definition as the bridge.

This activity involves a fair amount of interpretation, and is the strongest true modeling aspect of this task. We want to cluster X and Y as synonyms in the lexicon if they "mean the same thing" with respect to the domain. What are the various ways in which we might discover this lexical similarity? Here are a few scenarios:

- Suppose X and Y are different terms from different representative settings. How do we know they are related at all? By perceiving some kind of analogy across the settings.

- X and Y are the same terms used in different settings. Do they have the same semantics in these different settings?

- X and Y are different terms from the same *exemplar setting*. This means people in that setting see a distinction between X and Y. (They should also see a relation; e.g., "collapse family" and "collapse topic" are related terms from one exemplar setting.) If the distinction is relevant to the domain scope, the terms should remain separate in the lexicon. If the distinction is not relevant to the domain boundary, the terms could be gathered under one canonic term.

An example of clustering follows.

> *Example.* In the Outliner domain, many outliners provide some sorting functions; for example, an outline-style directory browser will allow sorting of the files and directories in various ways: by name, by date modified, etc. Should two distinct phrases "sort by name" and "sort by date" be entered in the lexicon?

> In this case, the recommendation would be to have a single term for "sort" in the lexicon. The distinction between the two kinds of sorting involve a feature that is outside the focus of the outliner domain. On the other hand, there will be variations in sorting operations that are relevant to the Outliner domain itself (e.g., sorting items in the entire outline might need to be distinguished from sorting sibling items at a given level).

➤ Check for homonyms

As a complementary activity to the search for synonyms, check for *homonyms*, the same literal term used with different meanings in different representative settings. (For simplicity, we will assume that the same term, used in the same setting, will have the same meaning, although we have probably all been in settings where this was not the case.) There are several cases to be handled here:

- The term appears to be unique to a single representative setting. In this case, it is wise to double-check that the term is referring to a concept within the domain scope. If it is, then it should either be synonymous with terms from other settings, or the representative data collected so far might be inadequate for comparison of the domain features referred to by this term.

- The term is used in several representative settings and its usage appears to be consistent with respect to the domain scope of concern. In this case it is not necessary to include multiple precedents for the term in the lexicon entry; it might be worth noting that the term has been verified in this way, however.

- The term is used in several settings and its usage appears to vary non-trivially. In this case we want to ensure that two separate entries are made in the lexicon. The same term, *with its different precedence* for each representative case, may thus be listed twice in the lexicon. However, the repeated terms will be in separate entries and the canonic terms selected for the two entries will be distinct. In this case, neither of the two homonymous terms should be promoted to canonic status, to avoid confusion.

➤ Cluster synonymous terms

Associate synonymous terms as conventional *synonyms*, abbreviations and *acronyms*, case, voice (e.g., active-passive) and plurality variants, etc., may be clustered together.

Another syntactic variant that is worth weeding out are *inverse expressions*, that is, cases where the same operation is described from different perspectives (a "half-full half-empty" scenario). This should be distinguished from true antonyms or oppositional terms.

➤ Select canonic terms

For each cluster of terms grouped as an entry, select the canonic term.

You can optionally document the rationale for selection of the canonic term. The main decision is whether to promote a domain native term or introduce a new, presumably neutral term as the canonic term. The tradeoffs are that the introduced terms will tend to be more abstract, more "computer science-ish" and hence will weaken the domain-specific "feel" of the vocabulary. On the other hand, it may often turn out that each candidate term carries strong system-specific connotations that you want to avoid. Apart from the grouping of the *synonyms*, this naming activity is another principal modeling aspect of this overall task.

➤ Convert features to canonic terms

Validate choices for canonic terms by converting a sampling of the FEATURES OF INTEREST to statements that utilize the canonic terms. In addition to the FEATURES OF INTERESTS, DOMAIN DEFINING RULES within the DOMAIN DEFINITION can also be canonified. This serves primarily as

a validation activity within this task. Later in the *Model Domain* phase FEATURES OF INTEREST will be more systematically modeled.

Verify that the set of canonic terms are rich enough to convey the distinctions made by domain practitioners. Be alert for terms occurring in the data that do not map clearly to a canonic lexicon term, but seem to have a distinct semantic sense within the domain. Add these to the DOMAIN LEXICON as needed. Also see whether the canonic terms are coherent when used in association to describe a given feature.

➤ Validate lexicon

Validate the DOMAIN LEXICON according to the following criteria:

- *Minimality.* All terms included in the lexicon have specific meaning within the domain settings. No general definitions of terms are intermixed with domain terms.

- *Compactness.* All synonyms within the lexicon have been identified and grouped within a single term cluster. No two canonic terms that are synonyms.

- *Coverage of models.* When updated later in the modeling process, the lexicon contains all new terms introduced by modelers within the DESCRIPTIVE MODELS.

- *Comprehensibility.* The lexicon should be comprehensible to informants without their needing to understand the modeling representation formalism.

➤ Use the lexicon

If possible, get the lexicon used while modeling proceeds. A good practical way to test it out is to hold a meeting with investigators, modelers (i.e., those who will be involved in the *Model Concepts* task and beyond), informants, and some other practitioners who have not been involved in developing the lexicon. The practitioners involved should be drawn from diverse settings within the domain scope. The technical topic for the meeting need not concern the further objectives of the domain engineering project directly. See if the DOMAIN LEXICON can be used effectively to translate the terminology used by one set of practitioners into terms understandable to the others, and vice versa. Check that the clustering of terms holds up in usage, that there are distinct terms for significantly different concepts in use, and that the canonic terms are reasonably acceptable.

➤ Maintain lexicon

As modeling proceeds, iteratively update the lexicon to serve as an ongoing snapshot of the modeling team's relative consensus on domain language. Extend the lexicon as needed to reflect to include terms selected by modelers. These terms will, in subsequent passes, often refer to concepts for which there are no distinct practitioners' terms available. In this way, the DOMAIN LEXICON remains a current snapshot of all the vocabulary introduced in the model, including terms borrowed from domain practitioners and new terms introduced by modelers to express subtle differentiations not required in informal practice.

When To Stop

An initial version of the DOMAIN LEXICON should be stabilized before moving on to the *Model Concepts* task. This initial pass can be considered finished when:

- Key terminology needed to facilitate data acquisition has been captured from artifacts and

informants from multiple representative systems and recorded in the DOMAIN LEXICON.

- Feedback from informants has been obtained. The feedback has validated the basic structure of the lexicon.

Since it serves as a pipeline between domain information in the DOMAIN DOSSIER and the DESCRIPTIVE MODELS, maintaining the DOMAIN LEXICON is also an ongoing task throughout the *Model Domain* phase.

Guidelines

- Do not model semantics in the lexicon. It is easy to blur the distinction between the lexicon structure and the deeper semantic modeling of concept and feature modeling activities. However, this will compromise the primary usefulness of the lexicon as a front end to these other models, and a means for communication with domain practitioners who have no background in the modeling formalism used by the project.

  Terms chosen by modelers may be filtered into the lexicon (e.g., concepts that represent subtle distinctions for which there is no clear terminology in the domain). If the lexicon is updated to reflect such taxonomically derived terms, do not allow this to alter its usefulness as a "flat" term space. A reader should not need to understand modeling semantics to find terms in the lexicon.

- Reflect practitioners' terminology. The lexicon should be confined to domain practitioners' language and the relationships they perceive as significant. Where semantic relationships recorded in the lexicon should be inferable from information sources, along with the terms themselves. Relationships as perceived by the modelers should be noted informally and separately, or deferred until the *Model Concepts* task.

6.2.2 Model Concepts

We have assembled and initially interpreted some set of domain information, recorded in the DOMAIN DOSSIER and FEATURES OF INTEREST. We have identified key DOMAIN TERMS within this material and have made initial choices for reconciling variant terminology in the DOMAIN LEXICON. This has been done within the context of an established DOMAIN DEFINITION, a continual reference point for what is and is not germane to the domain.
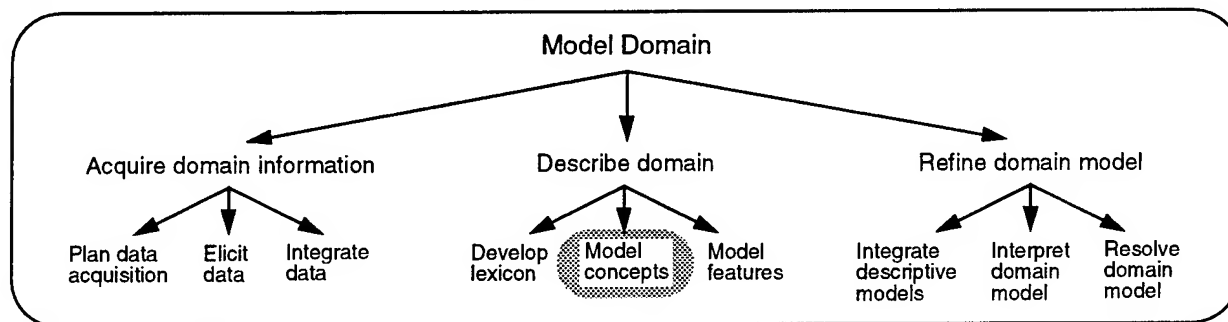
Exhibit 64. Model Concepts Process Tree

Our eventual goal is to synthesize all this data into a DOMAIN MODEL that will help software developers construct both systems and reusable assets in the domain of focus. To accomplish this, our goal is to create a language for consistent description of existing and envisioned domain capa-

bilities. In the *Model Concepts* task we specify the semantics for this language. *Model Concepts* can be considered, in many respects, as the central task of the ODM process model. Each prede-cessor task provide some essential data to inform the modeling of key concepts of the domain; all subsequent tasks follow implications that emerge from this modeling activity.

There are particular challenges in conceptual modeling in the domain engineering context. Main-taining focus and appropriate depth of detail is an ongoing challenge. Even with careful selection of representative data and the filtering of language through lexicon terms, it is easy to create a maze of conceptual terms and relationships that are difficult to critique, submit for review by informants, or build on in later steps. In addition, in software-intensive domains, distinctions between concepts as needed for domain modeling of variability across exemplars and entities used in the system modeling representations can be quite subtle and easily confused.

Approach

The Approach subsection for the *Describe Domain* sub-phase (Section 6.2) introduced the ODM notion of a domain *concept*. As with many other terms in this guidebook, we use a familiar term in a technical sense which, while consistent with informal usage, is restricted in specific ways. To understand the ODM approach to the concept modeling process, it will be helpful to take a closer look at how concepts are defined and used in the ODM context. We will then offer a fairly extended example of some concepts that might be useful for a particular domain.

Some Examples of Concepts

In the context of domain engineering in software-specific domains, a concept can refer to any arti-fact, workproduct, work process, area of knowledge or human practitioner within a domain set-ting. Concepts can describe both abstract elements of discourse and human interaction (design principles, issues, rules) or concrete entities. While we tend to think of entities primarily within the operational environment for the systems within the domain (the "usage setting") concepts can also address elements in the development setting or any other domain setting. For any domain, what we choose to focus on as concepts should reflect phenomena that display patterns of com-monality and variability across domain exemplars significant for our modeling objectives.

In practice, we almost always begin modeling with some pre-defined repertoire of conceptual cat-egories that are thought to be domain-independent and sufficiently comprehensive to capture the semantics of any given setting. Different disciplines, supporting methods and representations have their own versions of these *concept starter sets*. Most domain analysis methods embed par-ticular starter sets of this kind. In the ODM context, we consider concept starter sets as another component of the Concept Modeling supporting method area, described in Section 8.3.

To help ground the process descriptions for this task, we will present a particular concept starter set that evolved out of experience with the Outliner domain as an extended example.

The following bullets offer some suggestions for possible conceptual entities to be modeled. These are intended to be neither exhaustive nor prescriptive. It is assumed that each project will tailor this list to suit its own needs, and will discover other conceptual model types appropriate for its domain of focus. *It is important to emphasize that the particular conceptual categories pre-sented here should not be considered part of the core ODM method and process model.* They may not be appropriate for all domains and may overlap in arbitrary ways with categories used in other methods. They are included in the main section of this document for the purposes of illustration only, and to offer modelers a concrete starting point.

- *Objects* are often pivotal domain concepts, because they may represent key abstractions

around which domain operations are organized. The term is not intended to imply a strict object-oriented protocol, although object-oriented analysis techniques are one starting point for methods and representations. Some indicators of "object-hood" for a conceptual entity might be:

— correspondence with an agent or entity, a document or workproduct, an application or data item in some domain setting;

— the ability to create multiple instances of the entity as a domain function;

— the ability to separately manipulate sub-components of the entity; and

— the entity's participation in larger ensembles as a sub-component.

> *Example.* In the Outliner domain, different outline-based applications could be modeled as objects (Outline Application). Outline documents themselves could be modeled as objects, with their constituent sub-structure. In any given usage setting, there would be only one Outline Application executing (essentially functioning as a tool or process in the work setting) but there might be many outline documents being worked on. In a development setting, there might be multiple outline applications being produced.
>
> Entities within a given outline could also be modeled as objects. For example, different exemplar systems use various terms to describe outline headings. They may use distinct terms to indicate structural elements that can contain sub-structure (e.g., headings) versus elements that cannot contain sub-structure (e.g., "text").

In modeling objects, both taxonomic and structural (or, for larger-grained objects, architectural) relations could be modeled. Taxonomic relationships can be defined between terms for objects when the terms represent different degrees of knowledge about the objects or when a specialization applies to a subset of the objects categorized by the parent.

> *Example.* There may be distinct terms for "structure-bearing" elements in a particular outline that do or do not have sub-headings (e.g., "full" versus "empty" headings). At this level, the decision as to whether to model the concept "full heading" as an object or in some other way is part of the concept modeling task.

- *Operations*, functions, or behavior concepts form a complementary model to the objects model. Note that these concepts map most directly to the intuitive notion of features as user-visible functional capabilities.

> *Example.* Operations from the Outliner domain include terms such as "promote" and "demote", "expand" and "collapse", "create new heading" etc.

Semantic relationships to model among domain operations can include the following:

— *Specialization.* For example, "collapse heading" versus "collapse family" which acts recursively on all sub-headings within the outline.

— *Aggregation of compound operations.* For example, "create aunt" could be modeled as a composite of "move to parent" and "create sibling".

— *Attributes of operations.* e.g., outliner operations have an associated scope of effect ranging over "character", "word", "current heading text", "whole outline", or "selected region". (Such attributes of operations may be more appropriately modeled as features.)

Note that in these examples object terms are freely intermixed with operation terms. Opera-

tions will often not be fully factored out in raw domain data. There are advantages to factoring them out separately in the CONCEPT MODELS.[6]

If the domain of focus spanned the work practice setting of people using outliners to perform various writing tasks, then operations concepts could be extended to include activities or processes performed by people. This underscores why choice of domain settings is a key aspect of the *Focus Domain* task. Without this focus specified, the ontology of what is meant by a domain object or operation is entirely up for grabs.

- *Relations* are second-order conceptual entities based on objects and operations. There can be object-to-object, object-to-operation, and operation-to-operation relations.

  Example. In the Outliner Domain, the term "headings" denotes a basic object in the outline structure. The terms "sister" or "sibling" are synonymous terms denoting a particular relation between headings. The term "first," "last," "previous" etc. are positional terms that may be important elements in specifying the semantics and behavior of given domain capabilities.

- *States* or *Conditions* are other conceptual entities that build on the primitives like objects, operations, and relations. These may refer to states of objects, or overall conditions in the operational or usage environment for the system.

  These can be distinguished from the more familiar use of states within representations such as state transition diagrams, which are designed to facilitate specification of the behavior of a single system under a set of input conditions. A taxonomic model of conditions or states classifies significant variability in analogous condition and state descriptions across multiple representative systems. The model can in principle include sets of conditions not realizable by any single system.

  States have linkages to other types of conceptual entities and semantic relationships to other conditions. For example, one condition could be said to specialize another if it implies an increase in the information available about the state of the entity being described.

  Example. As we mentioned above, a heading may be in a condition of being "full" (having sub-structure) or "empty." This is a transient state that can be changed for a given object by the performance of some operation in the system setting. A heading that has sub-structure can also be in a "display state" where the substructure is revealed ("open") or hidden ("closed"). This state can change without structural changes to the outline itself being made.

- *Error values* often form an implicit taxonomy or classification scheme for unusual system conditions. Typically, system designers model such error conditions in a relatively flat error coding system with informal semantics. Some programming languages like Ada offer more semantic control over errors through exception-handling mechanisms, etc. Even in older languages, however, some taxonomic relations between error conditions can be deduced from error message text, conditional statements, error processing and user documentation.

  Of particular interest are errors that can be considered special cases of other error conditions, and understanding which errors take precedence when multiple error conditions occur simultaneously. These distinctions can lead to subtle variants in the behavior of components when lifted out of their original usage setting.

[6.] Having separate taxonomic models of objects and operations is stylistically a bit different than a strict object-oriented approach. However, this distinction is an attribute of the illustrated concept starter set and not an inherent aspect of the core ODM process.

Example. One difficulty in outliner applications is when deeply nested outline items need
to be displayed in the limited screen real-estate of a computer window. Curiously, hierar-
chical tree displays (a related domain) have the notion of an "unlimited window" that
can be vertically and horizontally scrolled. The text document legacy of most outliners,
combined with the assumption that outline items will include arbitrarily long paragraphs
of text (not tree nodes with short labels), leads to ugly "edge conditions."

Consider what happens in an outliner application where the limit of outline levels is set
to number n, and the user moves a subtree deeper in the outline in such a way as to
exceed this maximum. (If they directly tried to create a heading at level n+1, the system
would presumably prevent them.) Different systems handle this error condition in differ-
ent ways, and other (more elegant!) ways can be imagined that are not supported by any
current exemplars. One exemplar simply "squeezes" the errant headings to the last avail-
able level, so that headings from multiple levels get reduced to one level. A more desir-
able behavior might be for the system to prevent the repositioning that will have
destructive impact on the outline structure.

- *Constraints* introduce still a higher-order conceptual level in modeling. (As these levels
increase, the expressive power of any given *concept modeling representation* will be taxed
to varying degrees. For example, constraints might need to be expressed via production rules
or first-order logic, and may or may not be amenable to automated support.) A constraint can
be thought of as a relation between states.

    Example. As the behavior of outliner applications is more thoroughly studied, it becomes
    clear that certain systems obey the constraint that a heading at level n cannot directly
    own a sub-heading of anything other than level n+1. In other words, the outline structure
    cannot jump levels. Other applications do not enforce this constraint.

- *Domain-specific concept types.* In addition to the suggested categories above, each domain
includes areas of knowledge requiring unique concepts. While such concepts can be loosely
classified according to a standard taxonomy, they have important distinctive qualities that
must be considered as part of the modeling effort, and will often be crucial for the domain
engineering effort.

All of the concepts enumerated above address elements that could be considered within the
domain scope. In addition, it will frequently be the case that concepts are needed for certain con-
textual elements, i.e., common and variable characteristics of settings that have direct input on
domain elements of interest. Unlike states or conditions, which are more transitory and are used to
specify the behavioral semantics of individual entities, these contextual characteristics are more
likely to affect configuration of systems at a global level, i.e., characteristics of the setting itself.

For example, it may be useful to model *environmental characteristics*, the range of anticipated
variability in operational conditions factored into the design of a given system. Most system engi-
neering models describe the operational system itself. Assumptions about the environment in
which a system will operate are sometimes only informally documented in requirements. In engi-
neering for reuse, where multiple settings of operation are a given, potential variability across
these environments must be considered. To capture this contextual data, it is necessary to identify
aspects of requirements or system descriptions that encode information about the anticipated
characteristics of the world outside the system (e.g., timing requirements expressed as a minimum
or maximum interval to be handled between two external events driving a system). Such require-
ments indirectly state expectations about the environment; these expectations can be modeled and
compared with expectations embedded in the design of other exemplar systems.

An important caveat: just as the operations concepts may overlap into territory to be addressed more completely in the *Model Features* task, so the modeling of contextual elements of settings would ideally be thought of as part of the *Interpret Domain Model* task within the *Refine Domain Model* sub-phase. It can be very difficult to bound which aspects of a given setting need to be modeled explicitly as concepts. The setting serves as a focusing criterion for the domain, but the setting is not the domain itself. In the later interpretation task, we have a stable enough model of domain concepts and features to provide criteria for selecting and filtering relevant *setting characteristics* to be modeled. Therefore if such contextual concepts are captured in this task they are best thought of as previewing the setting characteristics that will be modeled more systematically in *Interpret Domain Model*.

> *Example.* In the Outliner domain, we will need to know the various system platforms supported by various exemplar systems; e.g., one is confined to the Macintosh environment, another to Windows, still another to Unix. Should we introduce a concept for "Computer System Platform"? Is this concept itself within the semantic scope of the outliner domain? In this case, we decide to note the concept but to treat it, later in the process, as a feature of outliner applications.

Workproducts

■ CONCEPTS OF INTEREST

A list of *concept areas* to be developed. Used by modelers in allocating modeling tasks, filtering data, and model development. Concepts used for a domain engineering model may range from concepts for concrete artifacts, such as code modules or other system artifacts, to more abstract conceptual entities such as types of errors, objects, or operations in the domain. The purpose of enumerating the concept areas to be developed are to make sure all essential topic areas are being addressed, to provide a basis for balancing the modeling effort spent, and to provide criteria for knowing when you have drifted out of range of relevant topics.

Some symmetries between the DOMAINS OF INTEREST and CONCEPTS OF INTEREST workproducts are useful to note. Neither workproduct is intended to be an exhaustive list. Both are selected with strong links to stakeholders (hence the "of interest" suffix) though this is not the exclusive source of criteria for selection. In neither case are relationships between items (domains or concepts, respectively) expected to be known completely or formally when the workproduct is first created.

Each concept area in the list is documented with the following elements:

- *Name*: a term or short phrase that captures the intended semantics of the concept area. Where possible this name should be drawn from the DOMAIN LEXICON. The name will eventually be used for some conceptual entity in a CONCEPT MODEL.

- *Derivation/Rationale*: Why are we interested in this concept area? In particular it is useful to know which concept areas were suggested by general (domain-independent) sources and which came out of examination of domain data. The Activities subsection below list a number of sources for candidate concept areas (e.g., supporting methods, project objectives, the domain definition rules, DOMAIN DOSSIER, etc.)

  We list this as two pieces of information: the derivation rule and an indication of the specific source involved. Note that a given rule might be applied to several concept areas and that any one concept area might be motivated by more than one derivation source.

- *Ontological Description*: What types of entities would serve as instances of the concept? The aim is not to pin down the extent or viewpoint of the concept right away; rather it is to record

the modelers' intent for each concept area. Clarifying this in the initial definition can help ensure consistency in interpretation and to avoid certain well-trodden modeling errors, such as mixing things with categories (wine bottles with vintages), or views with things (paper documents, files, internal data structures, interactive presentations). It is a lot easier to do this at this point than to sift through miles of "spaghetti models" later on!

For example, suppose one defined a model of Wines in terms of wine regions (e.g., Bordeaux). Another model might use the same term as the name of a generic category of wines, where particular vintages would be the individuals. Still another model could use the term to refer to individual bottles from a single vintage (e.g., for a quality control group). In each case, although the general subject is wine, the specific semantics of what an individual in the model stands for is different.

In software-intensive domains, ontological problems crop up around different settings and capabilities that cross setting boundaries. I.e.,does "object" mean an object in the user interface interaction, a data structure in run-time memory, an outline document, or a development artifact?

- *Illustration of entities*: Provide one or two prototypical examples/counterexamples or informal feature descriptions as a supplemental way to define the concept's ontology.

- *Informal relationships between concept areas*: We can't be certain about the relationships between these things at the starting point of concept modeling. Note suspected relationships, without trying to be formal, systematic or exhaustive.

An example of concepts of interests follows.

Example. For the outliner domain, we know that we are interested in having a model of the different kinds of outline structures that are supported by various applications. We decide that we want a model of such "outline data structures." In the CONCEPTS OF INTEREST list, we add the following entry:

— *Name*: "Outline Data Structure." Note that there is a fair amount of ambiguity in the name. It is really only a place-holder for now; it should be distinct from other concepts on the list, but the clarification comes in the subsequent elements.

— *Derivation*: Why are we interested in this aspect of the domain? One of our domain-specific PROJECT OBJECTIVES is to specify a document exchange format that will allow outline-formatted data to be exchanged across heterogenous programs. To meet this objective we will need a model of the structural elements and constraints on different "flavors" of outlines.

— We also are working from starter concept areas that involve looking for Objects, Operations, Relations, and Constraints. We apply this starter set to the setting of the outliner program in use and ask: "What objects are involved in the domain processes in this setting?" This leads us to the "Outline Data Structure" concept area from a different source. This is also noted in the list.

— *Ontological description*: Similarities and variations in the structure of outline documents and in the static components of these documents. We are *not* concerned here with being able to describe the detailed hierarchical structure of any particular outline document (i.e., we are not looking for a way of comparing specific documents). Nor are we comparing operations given applications might provide for viewing or manipulating outline structures. We are also not concerned, in this concept area, with the internal data representations of the outliner data. For example, a non-level-skipping-style outline could be

stored via a representation that did or did not support level-skipping. This distinction is not the focus for this concept area.

— *Illustration*: Some outlines can jump levels (i.e., a level-1 heading can have a direct sub-heading at level 3, etc.). In other outliners this configuration is impossible. Some outline structures have a "body" element that is owned by a particular structural level element, but cannot have sub-structure of its own; other outliners have only structural elements. These are examples of the kinds of conceptual distinctions of interest for this concept area.

■ CONCEPT MODELS

These are the specific models created to capture the semantics of the domain. Each concept model includes one or more of the CONCEPTS OF INTEREST as well as related concepts in those areas. In domain modeling, where multiple models may be developed by multiple modelers, lack of explicit shared concepts about the purpose and content of models can result in models of poor quality, inconsistent semantics and lack of modularity.

Concept areas will not necessarily correlate to names of separate models; neither is there any assumption that all concept areas will wind up in one model. Each concept area will eventually be modeled by one or more concepts, which will in turn find their way into models, not necessarily in a one-to-one correspondence. Some concepts will turn out to be related within one model; other models will be created that are not in the list.

There is little specific to be said about the structure and content of each CONCEPT MODEL. These aspects, and the representation strategy for the model, are determined by the Concept Modeling supporting method (or formalism) selected. Different models may invoke different formalisms.

■ NEW DOMAIN TERMS

Terms that are introduced as a result of the concept modeling activities for which there are not good existing terms in use by domain practitioners. These terms will generally reflect names of concepts that have been introduced in some CONCEPT MODEL. The terms are passed to the *Develop Lexicon* task to be added to the LEXICON MODEL.

When to Start

Selection of the CONCEPTS OF INTEREST can begin as soon as:

• Domain of focus has been selected and bounded. The *Bound Domain* task has produced initial DOMAIN DEFINING RULES and EXEMPLAR SYSTEMS SELECTION and these workproducts have been cross-validated for consistency. The concept areas selected should all fall within the scope of the selected domain (and will reflect a further focusing of attention within this domain boundary).

• Key interfaces to domain functionality have been identified in the DOMAIN INTERACTIONS.

Initiating this task before the above criteria are met raises several areas of risk for the *Model Concepts* task, analogous to the risks in the *Acquire Domain Information* sub-phase. Models unnecessary to the domain may be developed. Other key concepts, specific to the domain, may not be considered in the overall structure and relationships of the DESCRIPTIVE MODELS, and may be missed entirely or added later in an *ad hoc* manner.

Comparative modeling of commonality and variability within domain data in the *Model Concepts* task can begin when:

- The DOMAIN DOSSIER includes sufficient diversity of data to ensure that the DESCRIPTIVE MODELS developed are not skewed by a few special cases. Modeling should not begin on the basis of study of one system, nor be deferred until all systems are studied. A rough rule of thumb might be that, for a representative set of seven systems, three systems would constitute a good starting basis for modeling.

- The DOMAIN LEXICON is established, and procedures for migrating lexicon terms to the CONCEPT MODELS are in place. Models developed with uncontrolled vocabulary are difficult to correct afterward.

- Data acquisition should *not* be completed. If the start of modeling is delayed too long after the *Acquire Domain Information* sub-phase, comparative interpretations may filter into data gathering and be more difficult to sift out. In addition, some insights generated could be lost if data acquisition personnel are not involved with model development.

Inputs

- DOMAIN DEFINITION. Captures much of the commonality of the domain. This information will be elaborated into concepts that describe the variability in the domain.

  — DOMAIN DEFINING RULES are one source of candidate CONCEPTS OF INTEREST.

  — DOMAIN INTERACTIONS. Interfaces to structurally related areas of functionality may become one area for concept modeling, in order to clarify the specified range of variability the model encompasses.

  — DOMAIN CLASSIFICATION. Source for high-level concepts for conceptually related domains.

- DOMAIN DOSSIER. Source of the INFORMAL FEATURES, and other domain data resulting from the *Acquire Domain Data* sub-phase.

- DOMAIN LEXICON. Source of domain terminology to be migrated into the CONCEPT MODELS.

Controls

- PROJECT OBJECTIVES. Used to develop criteria or specific candidates for descriptive models of intrinsic interest and potential assets of interest.

  Domain Selection Rationale included may indicate potential assets of interest.

- PROJECT CONSTRAINTS. Constrains the extent of descriptive modeling, based on available resources, skill levels of the team, etc.

Activities

The following are the main high-level activities in this task:

- Select CONCEPTS OF INTEREST utilizing various sources for candidate concept areas;

- For each concept area within the CONCEPTS OF INTEREST:

— Cluster relevant exemplars and domain terms within relevant CONCEPTS OF INTEREST;

— Define semantic relationships among domain terms;

— Find and name new terms and categories as needed to express gradations of meaning for which there is no clear pre-existing vocabulary in the domain.

• Organize the terms into one or more CONCEPT MODELS, using the appropriate Concept Modeling supporting method to structure the semantic relations among the terms.

Sequencing would involve planning/tailoring activities out of core scope (e.g., team versus individual modeling, link between data acquisition and modeling, does one person work on multiple exemplars and built one model; etc.

This task involves an interface to Concept Modeling supporting methods. See Section 8.3 for more information about these supporting methods. The kinds of models developed here, the number of models and their interconnections, and the sequence of their development are all highly dependent on the choice of a Concept Modeling supporting method. Choices made in these areas will determine the actual representation formalisms used for the CONCEPT MODELS and the choice of the specific set of models to develop. Factors such as the semantic sophistication required of the modeling representation and the strategy for linking and/or keeping separate the conceptual from the structural or operational system models will vary widely from project to project, and will largely determine the specific format of the workproducts created and the detailed steps followed.

Nevertheless, there is a core structure to this task that provides a useful framework regardless of the supporting method(s) selected. The following activity descriptions focus on the process aspects of the *Model Concepts* task. They should be read in conjunction with Section 8.3.

These activities are discussed in more detail in the following paragraphs.

➤ Determine concepts of interest

Select CONCEPTS OF INTEREST via the various methods outlined in the following sub-activity descriptions. These can be considered a repertoire of alternative strategies for deriving the CONCEPTS OF INTEREST from the various information sources available, as listed below. The sub-activities can be performed iteratively, in parallel, and/or selectively.

Basic Concept Areas

Given the structure of the core ODM life cycle and the assumption that a software-related domain is selected, a few concept areas will almost always be worth at least considering as candidates. Perhaps the primary concept area is a concept embracing "the application," that is, the domain of focus system functionality. For a vertical domain, this would be the entire application (e.g., the domain of flight simulator systems, spreadsheet tools, etc.) For a horizontal "slice" domain, especially one that does not always appear as a discrete separate component of the surrounding application, it is a little harder to name what this concept area would entail: i.e., that subset of functionality within the domain scope, wherever it occurs within the structure of the exemplar applications.

Suppose PROJECT OBJECTIVES were simply to catalogue a set of existing systems within the domain scope for potential reuse "as is." To be useful, the catalogue model would have to clarify various selection factors so that people could pick exemplar applications best suited to their needs. At a minimum, this would require some classification of the main variants of the applications (with, eventually, features for discriminating among the

229

variants). This would require this concept area to be addressed, at a minimum.

Typically, this broad concept area will be broken out into a number of smaller areas: e.g., for various system operations, data structures, categories of usage, etc.

Concept Modeling Supporting Method

A Concept Modeling supporting method may provide a *concept starter set* for the CONCEPTS OF INTEREST. If you use a supporting method as a source consider the following issues:

— Make sure the supporting method is an appropriate choice for the domain.

— Screen the starter set for concepts relevant to the domain and the PROJECT OBJECTIVES.

— Interpret the starter set appropriately within each relevant domain setting.

Strategic Objectives

Scan the PROJECT OBJECTIVES, particularly the domain-specific objectives, looking for objectives that call for particular types of assets to be built. Do some "backwards reasoning" to map these potential assets of interest to concept areas that would need to be covered in descriptive modeling in order to support development of these assets. The objectives may suggest fairly directly that certain types of things need to be talked about in the model.

For example, suppose that for the domain selected there would be strong interest in a software requirements checklist. This will mean that:

— requirements data should be examined, and

— a descriptive model of the commonality and variability in requirements documents structure, etc. should be developed.

Note that you are not committing to building all (or any) the assets used in the rationale for focusing on particular concept areas. The worst that will happen is that you will model an area that later becomes irrelevant.

Domain Defining Rules

These rules state what is included in the domain by definition. Often such rules will involve a high-level concept to be differentiated in descriptive modeling. Each defining feature or rule will typically suggest the root of a CONCEPT MODEL expressive enough to capture the significant variation across the representative set with respect to that broad feature category.

> *Example.* For the outliner domain, the ability to expand and collapse headings is a defining characteristic of the domain. Different outliners will expand and collapse in different ways; we will probably be interested in these differences. So "expand" and "collapse" are likely to be concepts of interest.

The three sources above have the advantage of having focus or prioritization built in to the rationale for using them as sources. They are also all available before the *Acquire Domain Information* sub-phase. In theory, therefore, they could be used to filter the data gathered as well as the models to be developed. This is a stylistic preference between a closed versus open style of information gathering.

Other sources for CONCEPTS OF INTEREST. The remaining sources each require a fair amount

of interpretive filtering. They are inputs listed in order of increasing difficulty in directly mapping to CONCEPTS OF INTEREST:

Lexicon Terms

The DOMAIN LEXICON is available at this point for candidate concept areas. Some indicators that suggest terms of particular importance in the lexicon include the number of synonyms found across exemplars and the number of phrases including the term.

> *Example.* The term "heading" shows up in the phrases "previous heading," "next heading," "sub-heading" and so forth. This implies that there is a semantic clustering around this term of significance in the domain.

Domain Information Sources

FEATURES OF INTEREST. In studying the exemplar systems and interacting with informants, a list of FEATURES OF INTEREST has been distilled from the raw information sources. These features presumably have been examined as one source for the lexicon. However, the lexicon does not try to measure how often a given term occurs in features of interest. So going back to the features themselves will provide a different order of data.

DOMAIN DOSSIER. Similarly, making the DOMAIN DOSSIER accessible for ongoing review and inspection by the modelers remains an important part of this process. While the DOMAIN LEXICON serves as a normalization, filter terms alone cannot replace the primary sources of the data itself or insights about the data. Just as the lexicon doesn't list every occurrence of a term in the features list, so also the features list does not try to map each feature back to each exemplar system.

> *Example.* An informant may have mentioned the feature that MSWord allows you to create "improper" outlines that jump from heading level 1 to heading level 3. In *Elicit Data* there has been, as yet, no systematic step of checking all other exemplars for this feature (or lack of this feature) Or is it a bug? As we begin to build the model we may need to go back to the raw data to check our facts, fill in gaps, or explore further.

Legacy Models

Domain practitioners (e.g., expert developers) have usually done their own share of systematizing and theory-building about aspects of their domain, hence will have formed their own taxonomic views of key concepts in particular areas. Modelers must find a way to utilize these insights; if it was interesting enough for an informant to model, it is probably going to be interesting for the domain modelers to model it.

However, it is important to represent this knowledge in forms that help keep it distinct from the terminology introduced by the modeling team itself, without taking the knowledge uncritically as *the* model for the domain. The "same" conceptual data may be viewed from multiple perspectives. In the *Acquire Domain Information* sub-phase modelers have gathered this "native model" data. This data may now require separate concepts of interest (and possibly separate models) for distinct stakeholder views.

This data is handled as a ***legacy model***, that is, a domain model that comes from an informant's own interpretation. Handle this by including root concept(s) for the model as CONCEPTS OF INTEREST. This is one case where there will be an extant model associated with the concept, and a model formalism as well. By treating these models as separate entities we need not accept the informant's model as the basis of our own. eventually, though, we will

231

face the challenge of integrating alternative models of the same conceptual data in some way.

Domain Interfaces

In the *Define Domain* sub-phase of the *Plan Domain* phase, a number of DOMAIN INTERAC-TIONS have been documented where domain functionality intersects with functionality beyond the domain scope. Each one of these interactions may become an area where key concepts are needed. Interfaces to structurally related domains will need to be covered by some model to accommodate the variability in the interface. Typically these borderline concept areas will not have criteria for completeness and consistency as will those wholly within the domain scope. However, they are important holding bins to keep around in the modeling process, if only to help short-circuit unproductive vacillation around domain boundaries, an ever-present problem as modelers delve deeper into the differentiating characteristics of the domain.

Modelers' Experience

Last but not least, a most important ingredient to allow for is the priorities for concepts that "pop out" of the data elicitation process. This can have an ethnographic flavor, if the emphasis is on concepts that domain practitioners seemed to stress or find particularly crucial. In its most open form, this can't be listed as an input, since it deals with learning and experience that, for whatever reason, has not been documented. It is therefore dependent on either a team design that allows data elicitation staff to participate in modeling, or that brings people in the two roles together for information transfer and knowledge creation.

➤ Prioritize concept areas

In this step we look at the set of candidate concept areas as a whole and make final selections. This could include consolidating certain models into one, addressing gaps that become evident in the set as a whole, or eliminating certain concept areas as potentially too redundant.

Once candidate CONCEPTS OF INTEREST have been assembled, perform a cross-checking and validation step by using various input sources as screens on the concepts. In general, we want concepts that score according to more than one input source (e.g., objectives and defining rules should agree that a given concept is core to the domain and the interests of the stakeholders). This results in a filtered list of concepts of interest.

One helpful technique in prioritizing the concept areas is to employ a *center/periphery representation*. Prioritization via a single, linear list is difficult to apply when there are multiple, heterogenous motivating factors for the choices, as in this case. Create a single diagram, where each concept area is represented as a small or large circle arranged around a central point. The most important metric is the distance of a given concept area from the center. More peripheral concept areas are those perceived as being less crucial to the overall effort. Other relevant metrics that can be represented (or elicited from informal use of the representation) include perceived relative size of each area, overlapping, clustering, spatial orientation and conceptual distance.

Note that this is a similar technique to the classification of CANDIDATE EXEMPLAR SYSTEMS into core, borderline and counter-exemplars. However, here the emphasis is less on boundary-setting and more on directing focus within an already reasonably bounded area. Also, because key DOMAIN INTERACTIONS are one source for concept areas, this representation may direct attention to the periphery as well as the core areas.

➤ Document concepts

Fill out the final CONCEPTS OF INTEREST work product for all concept areas. Although modeling can be done in a variety of sequences, there is some value in selecting and documenting an initial set of concept areas as a whole before diving down into modeling any one concept in great depth. The idea is to establish some loose semantic relations between the various models as early as possible.

➤ Plan detailed modeling activities

With the CONCEPTS OF INTEREST list in place, you can plan the intended sequence for detailed modeling activities. You can initiate modeling activities on a concept-by-concept basis. In any but the smallest projects, modeling may involve coordination of multiple people working in parallel.

Detailed discussion of these planning issues is beyond the scope of the core ODM process model. However, we can identify the following kinds of tradeoffs in planning the sequence of modeling activities:

- How do we link modeling tasks to elicitation tasks? Do we have the same people do both? If so, we will get the benefit of the informal knowledge acquired; if not, we will be more certain that the data has been recorded in an explicit representation.

- Do we put people to the task of filtering data from multiple exemplars into one model (a *concept-driven* approach), filtering data from one information source to all relevant models (a *information-driven* approach) or balance these approaches in some way?

- How do we coordinate possible overlaps in the scopes of different models? Unlike system development, where modular decomposition supports independent development of units, domain models tend to have numerous relationships and interconnections. Without an overall structure linking the models to be developed, it is easy to work at cross-purposes.

In any case, the plan eventually directs one or more people to work on a specific concept area. For each such iteration the selected concept will be referred to as the *concept of focus*.

➤ Associate exemplars

Extend the illustrative examples documented in the CONCEPTS OF INTEREST by finding a small set of concept "exemplars" for each concept area. These can be instances drawn from the DOMAIN DOSSIER or hypothetical exemplars. Note that terms from the DOMAIN LEXICON have links to precedent for use of the term, but this use will typically be a reference rather than an occurrence. Also, only one such reference is required for the purposes of the DOMAIN LEXICON. For the concept, however, we want a sampling of a few exemplars at the same *ontological level* but with some variability.

> *Example.* In the Outliner domain, if I choose the concept "expand-heading" I will want to go, not to the DOMAIN LEXICON to find where the term was referenced (i.e., in a user manual) but to some setting in which the described entity really operates. In this case, I could be referring to an actual expand-heading operation invoked by a specific user in a given outliner session (perhaps as saved in a keystroke log file for later analysis). The ontology of the concept would then be referring to an operation performed by a user as part of a sequence or scenario of operations. On the other hand, we could be referring to the fact that certain outliners implement expand-heading in one way, other outliners do it in another way. The illustrative exemplars should have made clear which of these interpretations is intended.

233

➤ Cluster lexicon terms within concept areas

Search the DOMAIN LEXICON for terms that seem strongly related to the concept area. Work only with canonic terms from the DOMAIN LEXICON. This is one of the most important aspects of following the process model as given. The DOMAIN LEXICON serves as a filter on the terms that make their way into the CONCEPT MODELS.

In effect, we are treating the concept name as the name of a bin. Now we are throwing a bunch of terms from the DOMAIN LEXICON into the bin. Some lexicon terms might be at the same level as specific exemplars we found, but we are most interested in terms that seem as if they would fit under the concept of focus as a specialization.

Do not worry for now about sorting the terms, checking for duplicates, or modeling the terms' interrelationships. You may also find broader or *generalizing* terms. Since the CONCEPTS OF INTEREST need not map directly to roots of separate models, some may turn out to be branches (non-root, non-leaf concepts).

If we find what we believe to be a superordinate term we can add it to the cluster for the concept area. However, we need to check to see if it perhaps belongs in another concept area first. If not, we may need to change the CONCEPTS OF INTEREST list and make sure the more general term still matches all the interest rationale for that concept area.

➤ Establish semantic relationships between terms

The call to the Concept Modeling supporting method passes the concept of interest, the covered concepts, and the exemplars. What is returned from the supporting method is:

- a structuring of the concepts with respect to one or more semantic relationships (typically specialization with inheritance) into one or more models;

- discarded concepts (judged out of the scope of the resulting models);

- new concept slots or requirements for concepts for which there appear to be no current names.

This activity description assumes a scenario where different representations may be chosen for the sets of concepts emerging around each of the CONCEPTS OF INTEREST. However, there will typically be a limited repertoire of representation types to choose from on this model-by-model basis. Establish conventions as needed that extend the requirements of modeling representations and methods, and coordination plans ensuring that separately developed models will be able to be integrated.

➤ Populate models

Populating a model involves adding descriptions of exemplar systems or artifacts into the model as *instances* of the appropriate categories. This is distinct from adding a new category to the model. Evolving the domain model and modeling domain data are closely intertwined activities. Adding instances may require adjustments to the model structure, and thus can be considered part of the natural evolution and validation cycle for each model. Each model can be considered complete when all representative systems have been characterized with respect to the main category in the model, and significant differences are captured in the model.

Map exemplars to the most specific associated concept that is appropriate. Check the model for balance. If some exemplars only apply at the highest level, while others are classified several levels down, look for concepts to interpolate (at mid-levels) or extrapolate (down to lower levels).

Can you think of an exemplar for every concept? If some concepts have no exemplars, enrich the set by looking back to the DOMAIN DOSSIER. Adjust models for additions, omissions, deletions, and combinations of exemplars and concepts.

➤ Find or select names for new semantic categories

Introduce new names for slot" in the concept model where differentiation is required that is not provided by the initial set of terms discovered for the model. This requires first rechecking the DOMAIN DOSSIER and DOMAIN LEXICON to see if there are native terms that would fit. If there are not, a new term can be developed, and then fed back into the lexicon as a modeling team-developed term. Another option is to loop back for more data acquisition, using the model or questions derived from the model to do more collaborative modeling with domain practitioners.

➤ Map informal features into concepts

Working from the FEATURES OF INTEREST (particularly from comparative features) look for terms and conceptual categories used in describing the semantics and relations of analogous artifacts. Convert the feature description into canonic terms from the DOMAIN LEXICON. Decompose each feature into constituent conceptual elements as described above. Document the common and variant concepts in the appropriate CONCEPT MODELS.

➤ Analyze domain defining rules

Starting from the DOMAIN DEFINING RULES, factor each rule into constituent conceptual elements. Look in particular for elements that appear in multiple rules. Add a representative element to the corresponding CONCEPT MODEL.

This strategy will result in links from defining rules to fragmentary concepts in models. As more defining rules are analyzed in this way, more concept elements are introduced into each respective model. Periodically in the process, switch from analysis to inspecting and re-organizing the concepts allocated to each model, so that there are semantic connections established among all elements in a model.

➤ Complete models

Based on the differential information obtained from the various activities above, complete the models to express the range of commonality and variability observed in the data. These become the final CONCEPT MODELS. Completing the models will include deciding on boundaries between separate models and checking for overlaps, redundancies and gaps.

➤ Validate and verify models

As with any task that involves potential iteration and/or parallelism in creating a set of workproducts, there is some validation to be done on a per-model basis and some to be done on the set of models as a whole. Note, however, that because concept and feature modeling are complementary and closely related modeling activities, some validation is more appropriately deferred until after at least some modeling of features has been done. Also, note that the entire *Refine Domain Models* sub-phase serves a validation and verification role for the concept models created in this task.

Formally validate the CONCEPTS OF INTEREST using the following guidelines:

- *Validate the set of concept areas for completeness and economy.* Understand rationale for inclusion or exclusion of suggested concept areas.

- *Validate common understanding of concept area distinctions.* To validate that the members of the modeling team have common understanding of distinctions between various concepts specified, take a small set of sample data and have modelers independently allocate data to the appropriate concept areas. Resolve conflicts to ensure that the concept descriptions are consistent and understandable.

Validate individual CONCEPT MODELS using validation principles of the Concept Modeling supporting method(s) employed, and using the following guidelines:

- *Validate consistency and completeness of separately developed models.* Since the modeling task may be carried out by separate teams (either in sequence or in parallel) it is important to have a closure step that verifies the integrity of the models. This validation step can be compared to integration testing in conventional software development.

Empirical validation of conceptual modeling material can be quite difficult. Domain engineering in software-intensive domains presents particular challenges. Some issues and guidelines are discussed below.

- *Review models with selected domain informants.* At a minimum, try to have at least one informant that is a good candidate for providing feedback on conceptual or taxonomic data. (Ways of selecting such informants are dealt with in the Information Acquisition supporting method area described in Section 8.2.)

- *Use small excerpts from the models for validation.* It is difficult, even for experienced modelers, to give detailed feedback on large amounts of material. It is much easier to prepare a smaller portion of the model, possibly even filtered or trimmed down in various ways, and to have a clear idea of the *kind* of feedback desired.

When To Stop

An individual CONCEPT MODEL is complete when:

- Relevant variants of the concept have been categorized from across each system of the REPRESENTATIVE SYSTEMS SELECTION.

- The appropriate Concept Modeling supporting method/representation has been selected.

- Conceptual relationships among the variants have been documented in the method of choice.

The *Model Concepts* task as a whole is complete when:

- CONCEPT MODELS are complete with respect to the model types specified in the CONCEPTS OF INTEREST. This signifies that the overall priorities of modeling have been addressed.

- Lexicon terms have been allocated to conceptual categories.

- Constituent concepts in FEATURES OF INTEREST have been modeled.

Guidelines

- Allow for overlap across concept areas. Modelers may decide to deliberately model some data redundantly in multiple models in order to:

    — verify modeling data by independent/parallel development;

    — generate alternative views of model entities for uses by different audiences;

    — assist in integration of models at a later phase; or

    — incorporate conceptual material directly from domain informants as separate models.

- Beware of confusion between concept models and system models. CONCEPT MODELS differ from functional models for a specific system, such as a data-flow diagram or a state transition diagram. These latter types of models are considered *artifacts* in ODM terms, because they describe the behavior of a single system, rather than commonality and variability across systems. Artifacts are examined (or sometimes generated) in the *Elicit Data* task.

- Beware confusion over domain settings in which a concept applies. A patient and a patient record are closely related but not identical. A software component may be modeled as an artifact in the development setting, as a process in the operational setting in which it executes. These complications are particular to the fact that conceptual modeling is being applied to the general domain of software development.

- Taxonomic data representations can be easily misinterpreted. Software engineers are accustomed to seeing hierarchical data, but will often interpret it in a systems engineering rather than a domain engineering context. Non-software oriented informants (e.g., end-users of applications) may also interpret domain model representations in a variety of ways. Pick the representation used for validation of concept models with the specific *validation audience* in mind; do not assume that the model representations most appropriate for ongoing modeling and asset base engineering will be able to serve "as is" for getting feedback on the models from domain practitioners.

- Ground semantic data with clear examples. Even for other modelers on the team, it can be difficult to look through large models of conceptual or categorical data. The rationale for why certain terms are needed in the models (especially new terms introduced by the modelers) may not be very evident.

    It's therefore a good idea to include plenty of simple examples with the model for the benefit of other model readers. No matter what representation is used, have a commenting capability available. For terms that are structurally related (i.e., nomenclature for different structural elements in the domain) some diagrams can help.

    This guideline may appear to conflict with the principle of not letting model terms themselves be too closely tied to one particular exemplar. But it is fine to include the examples you were actually thinking of as commentary on the models as they are developed. This mapping back to specific examples could occur in comments or annotations of the CONCEPT MODELS, via automated support that links the models back to the DOMAIN LEXICON, or in meetings where the models are reviewed.

    Example. In the Outliner domain, outlining text processors (like Macrophage Wort) often use the term "heading" for structure-bearing outline items. File system browsers like the Quince System 17 Seeker use the word "folder." From the standpoint of the concept models, these are both examples of "structural elements" of the outline data structure. "Heading" and

"folder" would remain as terms in the DOMAIN LEXICON and "Structural Element" would become the canonic term which appears in the CONCEPT MODELS. For validation, it might still be helpful to use these examples when eliciting feedback on the conceptual information. State that "Outlines have structural elements, including containers for content and other structural elements, such as folders in a file system browser, or headings in an outline-structured document."

- <u>Use multiple, modular models where possible</u>. The activity descriptions above may appear assume a scenario where multiple models are developed to encompass the various concepts of interest defined for the domain. In theory, one could address a domain so tightly focused that a single concept model is sufficient. The important idea here is that a degree of *modularity* in concept models can be as helpful as the same principle in programming. Multiple models can have the following benefits:

  — Enable more flexible team modeling strategies;

  — Allow for modeling of given concept areas from multiple modelers' perspectives, such as models developed by domain practitioners (so-called *native models*) as distinct from those developed by the modeling team; and

  — Help keep the ontological focus of each model more clear.

- <u>Consider hidden design assumptions in model partitioning</u>. Partitioning conceptual entities into categories such as objects versus operations already embeds significant design decisions. Different representative systems may represent conceptual entities in diverse ways that suggest allocation to different models. Capturing the range of such representations for exemplars studied will present significant challenges to the modeler's skill and will stress the representation formalisms being used as well.

  Example. In the Outliner domain, a system could implement an "outline presentation format" as an object, encapsulating a set of formatting decisions (heading fonts, etc.) that otherwise must be implemented as a series of individual operations.

- Object-oriented analysis techniques can be used in concept modeling, with the following caveats:

  — ODM neither assumes nor requires use of object-oriented techniques.

  — Taxonomic relations (i.e., specialization with inheritance) on object categories have slightly different meaning in the domain modeling context than in the object-oriented context. Specialization among objects (as well as for other conceptual entities) has definitional import only, not operational import (e.g., code-sharing, etc.).

  — Object-oriented principles such as encapsulation of methods with data, or run-time polymorphism should not be imposed if the exemplars studied do not exhibit this organization. Such restructuring would defeat the rationale behind descriptive modeling, unless done as reverse engineering to facilitate comparison of analogous artifacts and concepts.

6.2.3 Model Features

Our gradual construction of a *domain language* in the DOMAIN LEXICON has provided us with an initial controlled vocabulary. The CONCEPT MODELS produced in the last task defined the semantics of domain terms in focused *concept areas of interest*. In a sense, the CONCEPT MODELS tell us *what* is in the domain, and provide a sufficiently rich set of terms for those domain entities that differ in significant ways.
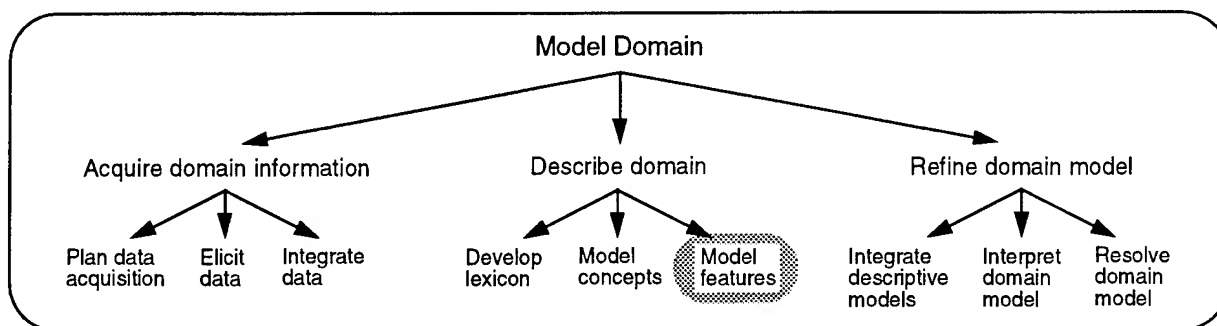
Exhibit 65. Model Features Process Tree

The primary purpose of the *Model Features* task is to enrich the CONCEPT MODELS enough to accommodate all variability in the domain, as evidenced by data culled from the domain's REPRESENTATIVE SYSTEMS SELECTION. Specifically, we want to understand *how* concepts differ.

There are several challenges in feature modeling:

- *Keeping concept and feature modeling activities distinct.* If you do not do this, you risk losing focus in the modeling effort.

- *Determining which features are worth modeling.* In principle, there are an infinite number of features that could be observed for any phenomenon. We need clear criteria for filtering what ought to be in the *feature model*.

- *Modeling concepts and features in a common representation.* Features can be modeled in very simple and straightforward ways. More semantically rich feature models create certain formal challenges that arise in modeling concepts and features in a common framework.

Approach

Structure of a Feature

The term "feature" is really short-hand for a number of related constituent elements, that link features to other elements dealt with in the ODM life cycle:

- *Concepts.* A feature is defined with respect to a particular concept model. If the feature holds for every exemplar covered by a given concept, it is a defining feature for that concept; otherwise it differentiates the concepts in that model in some way. Features are usually structured in terms of feature categories and feature values; this simple scheme can expand to a full taxonomic model if appropriate for the feature in question.

- *Settings.* In the ODM context, concepts are always situated in a given domain setting. For example, the concept "outline" must be unambiguously defined with respect to the setting intended: i.e., as a data structure manipulated by a program, or a document being edited by a program user. Features must also be linked to the setting in which they matter.

- *Practitioners.* Features also need to matter to someone, a domain practitioner or more generally a domain stakeholder.

Features can be related to the concepts that they differentiate in various ways:

- *Properties.* A feature can be thought of as an assertion or statement that can be made about an

239

entity (an exemplar of a concept).

- *Components*. Structures that are components or sub-structures of other structures can be considered, in some circumstances, features of that structure.

- *Relationships*. Relationships between concepts can also be treated as features. The component relationship alluded to above could be considered a special case of this.

An Extended Example

To further clarify the relations between concepts and features, the different aspects captured by features, the relation of features to settings, we offer an example in some detail.

Example. One thing that characterizes outlining interfaces is that the heading display must convey a certain amount of structural information as well as the content information (e.g., the text associated with the heading). This structural information typically will include:

— Open/closed state of the heading: whether the substructure below the heading is being displayed;

— Empty/full (i.e., non-empty) state of the heading: whether there exists any substructure below the heading.

The interaction of 1) and 2) can be seen in that users may need to be able to distinguish between a full but closed heading (with hidden structure) and an empty heading.

— Exemplar A (a text editor) headings show a "+" sign when there is substructure, a "-" sign when there is no sub-structure. The "+" sign does not change on toggling the view open or closed.

— Exemplar B (a directory browser) uses a triangular icon to distinguish directories as opposed to files. The triangle toggles between a horizontal (indicating closed) and vertical (indicating open) position. There is no specific indication of whether the folder concerned is or is not empty.

— Exemplar C (a later-to-market text editor directly competing with A) uses a diamond icon with three states: "full and open" (a black diamond) "full and closed" (a grey diamond) and "empty" (a white diamond).

In the *Model Concepts* task, we have introduced a concept called "Heading" which denotes the actual heading data item in a specific outline. One property of a heading is its "content state": empty or full. Another property of a heading is its display state: open or closed. A given heading must have a defined value for each of these states. There are various ways to model this, depending on which property is seen as primary, as shown in Exhibit 66. For example, we could choose to model these states as direct specializations of the concept Heading, as in Figures A1 and B1. Or we could model the states as their own concept, as in Figures A2 and B2.

Figures A2 and B2 represent, at a simplistic level, a feature model for the concept Heading. That is, they describe properties w/ varying values that can be asserted about the concept of which they are features. (Since the content state is dependent on the presence or absence of substructure in the outline, we could also model that state by capturing the structural or component aspect as in Figure A3. This would be a component-style feature; the distinctions are not essential for the main point.)

A1. As taxonomy

has content state (1 .. 1)

has substructure (0 .. inf)

A2. As property

A3. As component
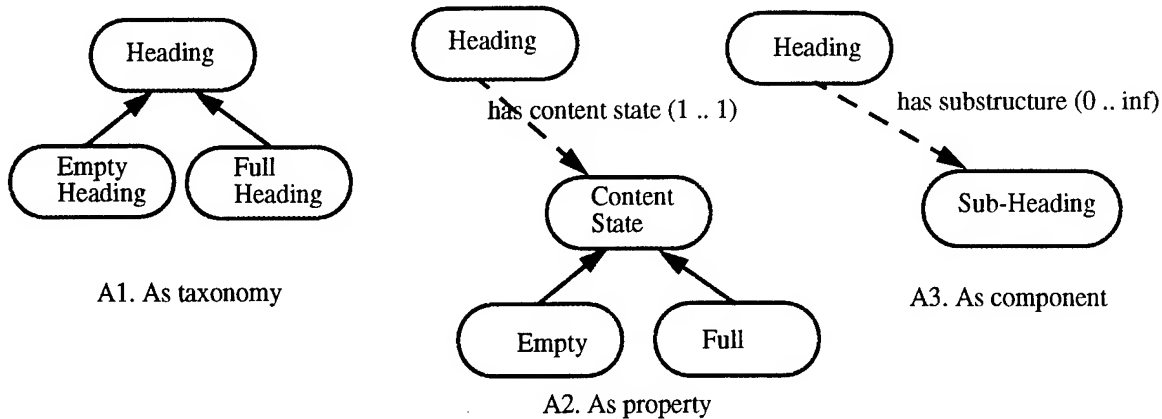
A. Alternative Ways of Modeling Heading Content State

has display state (1 .. 1)

B1. As taxonomy

B2. As property
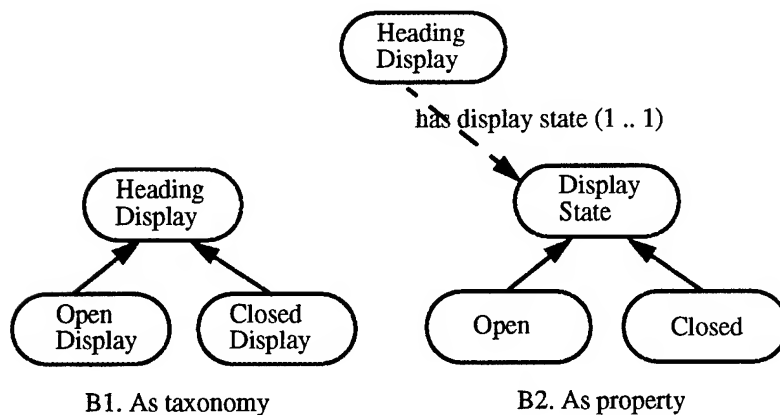
B. Alternative Ways of Modeling Heading Display State

Exhibit 66. Example: "Heading" Concept Model possibilities

Note that the Heading concept describes an entity that lives in the usage setting, more specifically, the operational environment for the application; i.e., a heading is part of an actual document being outlined by a user. Thus the features are associated with the usage setting.

A given heading can be in only one display state and one content state at any one time (or only one composite state determined by the two variant features open/closed, full/empty). A convenient way to represent this is with a two-axis "quadrant style" matrix, one of the simplest and most useful feature model representations. In this case, with two sets of two contrasting feature values yielding four possible states, the representation is quite reasonable. An example is shown in Exhibit 66.

For the purposes of domain modeling, however, we need to make an important shift: from the features of headings in the usage setting, to the features of outliner *applications* themselves. The interfaces of outliner applications differ in terms of the heading states they can distinguish in their display. Thus one important feature of an outline application will be the *set* of heading states it makes explicitly distinguishable in its interface.

241

```
                    ┌──────┬──────┐
           OPEN     │      │      │
  Content           │      │      │
  State             ├──────┼──────┤
           CLOSED   │      │      │
                    │      │      │
                    └──────┴──────┘
                    EMPTY  FULL
```

Display State
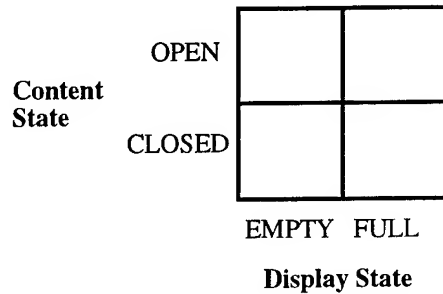
Exhibit 67. Example: A Simple Heading State Feature Model

We have three contrasting exemplars, each of which provides an interface which distinguishes some states explicitly, others implicitly, others not at all. We want to characterize outline applications in terms of *sets of states* they make visible. An outliner program interface could, in theory, support any subset of these (a total of 16 combinations).

To represent this, we could reuse the simple feature models A2 and B2. However, we now must reinterpret their semantics, or rather the semantics of their relationship to the relevant concept model. As related to the concept Outliner Application, the feature modeled by A2 will capture the assertion: "This outliner application makes this *set* of heading states explicitly visible in the interface." The same would be true for model B2 as a feature model. This ability to reuse models as different "views" of shared semantics applied in different settings is a major reason why ODM is oriented towards modeling via a collection of small models rather than one monolithic domain model.

interface distinguishes (0 .. 2)   interface distinguishes (0 .. 2)
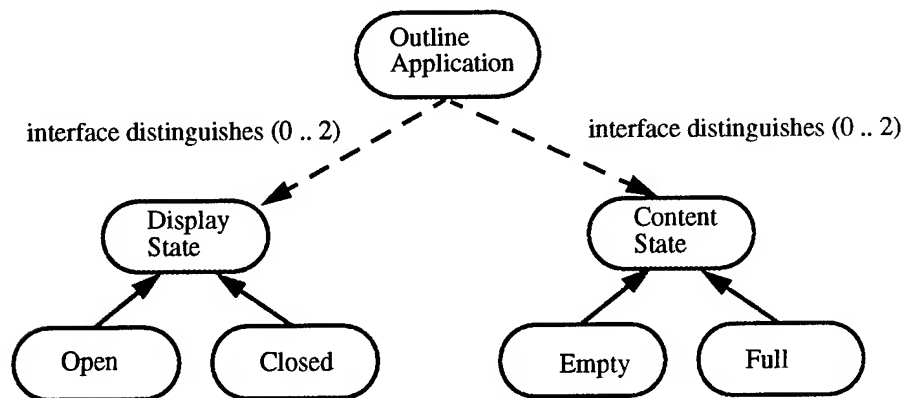
Exhibit 68. Example: Outliner Application Feature Model

As an alternative, we could extend the quadrant-style model in Exhibit 66 to display the sets of states supported by a given outliner. In this representation (customized for the domain), we indicate the number of states distinguished *by distinct icons* in the application via the number of X's. The placement of the X's shows the certainty of the state information revealed; borderline placement means the interface doesn't directly distinguish the two states.

This representation has the benefit of revealing some potential "don't care" conditions; but it turns out that some of these are "might care" conditions, depending on the setting in which the application will be used:

242

OPEN

Content State

CLOSED

	X	X
X X	X	X

EMPTY   FULL   EMPTY   FULL   EMPTY   FULL
Display State   Display State   Display State

Exemplar A   **Exemplar B**   **Exemplar C**
(text editor)   **(directory browser)**   **(newer editor)**
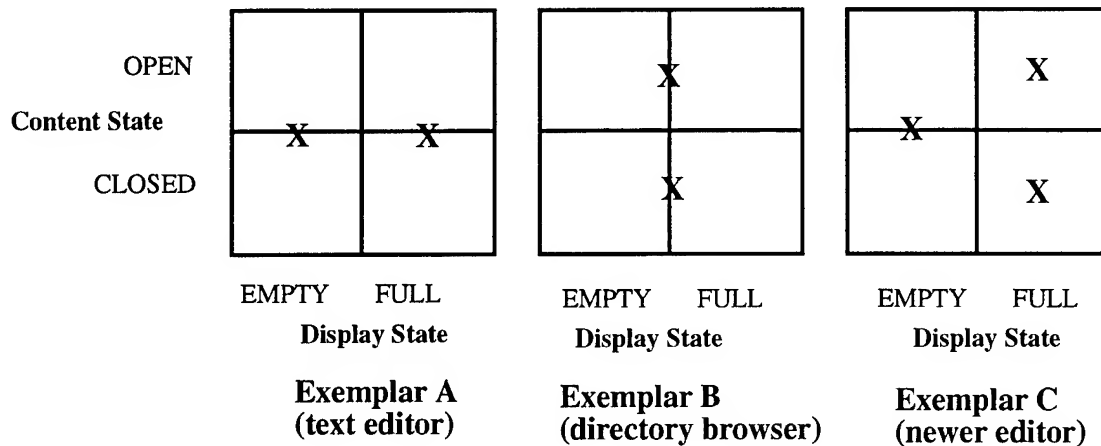
Exhibit 69. Example: Display State Feature Model and Profiles

— Exemplar A does not distinguish Empty/Open and Empty/Closed, which are debatably indistinguishable under most sequences of operations. Exemplar A also does not distinguish Full/Open vs. Full/Closed. This is usually evident by visual inspection for a document; but not always, e.g., if the heading sub-structure is scrolled off the edge of the window.

— Exemplar B does not distinguish either Full/Open vs. Empty/Open (again, typically but not always visible by inspection) or Full/Closed and Empty/Closed. The latter creates the inconvenience of needing to open folders to determine if they are empty or not. Presumably empty folders are considered exceptional conditions in file systems, which may have influenced the design choices.

— Exemplar C appears to have made an economical choice of features, perhaps in competitive response to the features already in the marketplace.

Such assumptions will be surfaced and validated later, in *Interpret Domain Model*. The main point of this second representation is to emphasize that feature models do not have to be represented in taxonomic form, and can be tailored to the particular characteristics of the domain and the intended usage of the models (e.g., for validation, automated support, etc.)

Workproducts

■ FEATURE MODELS

Models of distinguishing characteristics of specific artifacts or representative systems as a whole. Each feature is an assertion or statement that holds for the associated system entity and indicates a significant differentiation from the standpoint of some domain practitioner.

Each feature model maintains the following linkages:

• The stakeholder(s) for whom this feature is relevant or "of interest." This will usually be a domain practitioner in some domain setting (e.g., a system user in the usage setting, a software implementor in the development setting.)

• The setting in which the feature is relevant. This setting is usually strongly correlated with

the relevant stakeholders, as above.

- The *feature category* that encompasses the observed variants.

- The *feature values* that represent the variability with respect to that feature within the domain scope.

■ NEW DOMAIN TERMS

As with the new terms created in the *Model Concepts* task, these are terms that are introduced as a result of the modeling process to be passed to the *Develop Lexicon* task to be added to the LEXICON MODEL.

When to Start

- The *Acquire Domain Information* sub-phase has produced some informal features for exemplar systems to be compared.

- The *Develop Lexicon* task has produced some lexicon terms that can be used as the basis for the language used in feature statements.

The *Model Features* task can begin before CONCEPT MODELS are complete. In particular, features can be defined based on a single CONCEPT MODEL and it is also possible to begin modeling features directly. Trade-offs in the sequencing of concept and feature modeling are discussed in the Guidelines sub-section for the *Describe Domain* sub-phase.

Inputs

- DOMAIN DEFINITION. DOMAIN DEFINING RULES are one source for high-level features. DOMAIN INTERACTIONS may also become the sources for certain feature "views."

- DOMAIN DOSSIER. Another source for features. Not all relevant information will have been extracted as FEATURES OF INTEREST (see below); working backward from the CONCEPT MODELS may reveal additional features.

- FEATURES OF INTEREST. Extracted from informant interviews, artifact descriptions and comparisons. Features extracted from secondary domain artifacts (e.g., survey articles or interviews with a domain expert) may map most directly to *formal features*. Such sources of information will often include categorization of variation within the domain, without attribution to specific exemplar systems.

- CONCEPT MODELS. Provide the distinguished concepts to be differentiated by features.

Controls

- PROJECT OBJECTIVES. The intended uses and audience for the model will impact the level of detail for the features added to the model.

- CONCEPTS OF INTEREST. These are an important focusing device that control how broadly and deeply to model features. The features "grow" in the periphery around the CONCEPTS OF INTEREST. The CONCEPT MODELS provide the actual concept data, but the CONCEPTS OF INTEREST provide the prioritization.

Note that PROJECT CONSTRAINTS are not listed as a direct control here. These should influence the selection and scoping of the CONCEPT MODELS. The criteria for closure of the FEATURE MODELS are based on these. If effort in this task appears to exceed PROJECT CONSTRAINTS this implies that the corrective iteration should go back to at least the *Model Concepts* task.

Activities

Like the predecessor task, *Model Concepts*, this task involves an interface to Concept Modeling supporting methods. These are described in Section 8.3. The constraints on the supporting methods invoked here, however, are different than for *Model Concepts*. In that task, we suggested that it was inherent to the process that at least some aspect of the CONCEPT MODELS involve taxonomic modeling. FEATURE MODELS extend these models by providing *explanations* for taxonomic differences. These explanations can take the form of separate models in their own right, but these models need not be taxonomic in nature.

The following activity descriptions are a repertoire of alternative strategies for deriving FEATURE MODELS from various information sources available. They focus on the *process* aspects of the *Model Features* task. The activities can be performed iteratively, in parallel, and/or selectively.

➤ Extract features from rules

Translate DOMAIN DEFINING RULES into features. Every domain defining rule reflects an assertion that holds for all domain exemplars. Note the rule need not directly correspond to the assertion. For example, it could be an exclusion rule in which case it holds for counter-exemplars. In effect, such rules capture a common principle or attribute of domain exemplars. In this activity, this general statement is specialized into various cases that hold for some, not all, the exemplars covered by the domain defining rule.

➤ Formalize features

Working from the FEATURES OF INTEREST, translate each informal feature that involves a comparison between two exemplars into data reflected in the selected formalism. This activity can basically be seen as a *formalizing* step. The steps involved are as follows:

- Translate the feature statement into the relevant terms in the DOMAIN LEXICON.

- For features used to derive concepts in the CONCEPT MODELS, create a feature category that can be used to differentiate concepts in the respective models. For features not yet considered during the *Model Concepts* task, identify the concept(s) to which this feature relates.

  If the feature is considered an assertion then concepts can play the roles of either subjects or objects of these assertions.

One benefit of this activity is that features deemed relevant by domain practitioners will be systematically considered and reflected in the model. One risk is that a large set of features may be obtained, with no immediate principle for factoring these features into common models. The same kind of feature may have emerged in several different observed FEATURES OF INTEREST.

➤ Derive features to differentiate concepts

In this activity, working from the CONCEPT MODELS, we look for differentiating features to justify the concepts that have been introduced. Each model includes taxonomic data capturing the semantics of domain terms. Recall that concepts might have been introduced because they are

native terms, distinctions made by domain practitioners. Or they may reflect distinctions that arose in the course of comparative modeling. For each concept that specializes another concept, we want to know:

- *How* are the concepts different?

- *Why* is this difference relevant to the objectives of the model?

Depending on the specific model and the nature of the formalism employed, it may not be necessary to "decorate" each separate concept with a feature distinction. Any specialization should at a minimum include some restrictions on its inherited relationship links. But ideally it would have some new, local relationship that would point off to a concept in (usually) another model that is serving as a feature model. This local relationship, essentially, captures the *relevance* of the concept; why is it a useful concept to distinguish?

➤ Validate with representatives

In this activity, we check for the adequacy of a given model of concepts and features together, with respect to a particular representative. The resulting model is **adequate** with respect to the REPRESENTATIVE SYSTEMS SELECTION when there is a *feature variant* within the model that corresponds to the distinct characteristics of each representative system with respect to that feature category. When each key domain-specific term or phrase in an informal feature statement is linked to elements in the CONCEPT MODELS, the feature statement is said to be a *formal feature*.

The representative, the "subject" of the model, may not be the representative system in its entirety. It could be a specific artifact; for example, one could model features of outlines as data structures, or features of specific components of outliners. To complete integration, however, some profile for the representative system as a whole (i.e., the broadest scope of the domain within overall system functionality) should be developed.

Working from exemplars for a given CONCEPT MODEL, see if the model together with its associated feature models serve to adequately differentiate the examples. The important term here is *adequately differentiate*. This can be defined in terms of a specific set of representatives because such sets must be finite. A feature with respect to a set of exemplars must follow one of these cases:

- *Out of scope*: the feature is not relevant to any of the exemplars;

- *Defining*: the feature is common to all exemplars and reflects an essential, defining property of the domain. These features properly belong as part of the DOMAIN DEFINITION although it is not uncommon for new features of this sort to be discovered at this stage of the process.

- *Common*: the feature is common to all exemplars but reflects an accidental shared property of the set. This in turn could have two implications:

  — The feature is not relevant to the domain scope.

  — The feature is relevant, but the fact that it is common to all exemplars is a consequence of the sampling strategy used. In this case, a new exemplar should probably be added that has a different value for the feature in question. This would make the feature a true differentiating feature as below.

- *Differentiating*: the feature has different values for different exemplars. These are the kinds of features we are most interested in capturing in this task.

- *Anomalous/unique*: These types of features are quite similar to differentiating, except that they apply to only one exemplar of the set. They represent the "danger zone" in feature modeling, because it is easy for features that are not relevant to the domain to creep into the model in this guise.

These criteria serve both as a validation and an exit criterion.

When To Stop

You can consider any given FEATURE MODEL as complete when the features defined serve to differentiate the related concepts sufficiently to cover the cases within the domain scope. Thus there are a finite number of features required for a finite set of exemplars, since eventually each exemplar is differentiated by the features into a class by itself. Any further differentiation is basically adding detail about the exemplar that may not be needed for the purposes of domain modeling.

The *Model Features* task as a whole is complete when:

- There is sufficient "feature" coverage for all of the CONCEPT MODELS. Note that by linking back to the CONCEPT MODELS, closure criteria are easier to determine that by tracing back directly to exemplar data.

- The FEATURE MODELS as a whole provide a language capable of expressing key differences across the REPRESENTATIVE SYSTEMS SELECTION. Key differences are those considered significant to domain stakeholders, as represented by those informants consulted during the *Acquire Domain Information* sub-phase.

- Each of the REPRESENTATIVE SYSTEMS SELECTION has been examined according to the DATA ACQUISITION PLAN. The resulting data has been accounted for in one or more of the FEATURE MODELS.

- Each defining rule of the INTENSIONAL DOMAIN DEFINITION has incorporated in one of the DESCRIPTIVE MODELS. The model contains differentiating features for that category sufficient to describe variants observed within the REPRESENTATIVE SYSTEMS SELECTION.

The CONCEPT MODELS and FEATURE MODELS only need to note the commonality and variability across REPRESENTATIVE SYSTEMS SELECTION. They do not need to explain the rationale for the commonality and variability. This will be captured more systematically in the *Interpret Domain Model* task.

Guidelines

- <u>Extract features from requirements</u>. Requirements specifications are an important software life cycle artifact that can be examined during the *Acquire Domain Information* sub-phase. Software requirements have a special and close relation to features because, like features, they are comparable to assertions made about the systems to which they apply. A feature of a given system can be interpreted as the capability of that system to respond to a given requirement (or set of requirements). Thus, there is a useful duality between requirements and features.

  This duality can lead to confusion, however; therefore, it is useful to keep in mind ways in which requirements are distinct from features, including the following:

  — Not all requirements for a system become features of that system, e.g., they may not have been successfully implemented.

— Not all requirements are relevant to the domain focus.

— Not all domain features are likely to have been specified as explicit requirements.

- <u>Reflect practitioners' key issues in features</u>. In most domains there will be major issues that are the concern of practitioners

- <u>Preserve but isolate innovative features</u>. Processes throughout the ODM life cycle are intended to spur innovative thinking about domain functionality in a managed way. Innovative ideas for features will therefore be a natural by-product of this task. Features that seem relevant to the domain, but for which there is no precedent in exemplar systems, can be handled in two ways:

  — Expand the EXEMPLAR SYSTEMS SELECTION to include new systems exhibiting the features at issue; this may in turn necessitate adjusting the REPRESENTATIVE SYSTEMS SELECTION.

  — Document innovative features separately from precedented features, as input to the final task of the *Model Domain* phase, *Resolve Domain Model.*

6.3 Refine Domain Model

The final output of the previous sub-phase, *Develop Descriptive Models*, is a set of individual FEATURE MODELS that, taken as a whole, provide a descriptive language for common and variant features within the REPRESENTATIVE SYSTEMS SELECTION. This set of purely DESCRIPTIVE MODELS is a necessary but not sufficient basis for selecting a set of features to implement in the *Engineer Asset Base* phase. The *Refine Domain Model* sub-phase of *Model Domain* serves as the transition from descriptive modeling in the *Describe Domain* sub-phase to prescriptive modeling in the *Engineer Asset Base* phase.
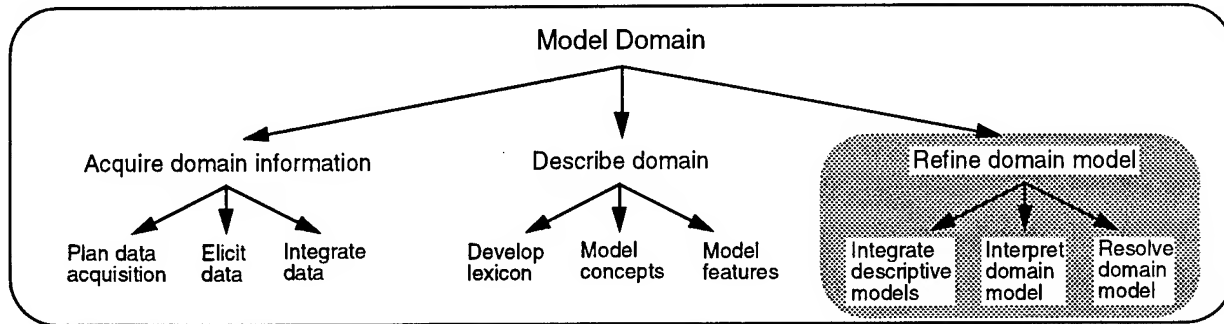
Exhibit 70. Refine Domain Model Process Tree

The primary goals of the *Refine Domain Model* sub-phase of *Model Domain* are to:

- Validate the DESCRIPTIVE MODELS created so far;

- Build a consistent *theory* or explanatory model for the domain that explains constraints and co-occurrences of features within domain exemplar systems (i.e.,*why* different exemplars have the particular sets of features observed.) This supports creation of an ASSET BASE where domain resources are made available to people with new requirements, in new (but related) settings. The explanatory model will provide a basis for selecting features appropriate for a given *market* or niche of customer settings.

- Understand essential versus accidental relationships between domain features so that resources can be reengineered and reconfigured in flexible and adaptable ways. The explanatory model also provides a foundation for exploring innovative combinations and restrictions of features that improve the overall coherence of the resulting DOMAIN MODEL.

There are a number of challenges in attaining these goals. A certain number of multiple viewpoints have been gathered in the *Describe Domain* sub-phase. These multiple viewpoints must be somehow reconciled, without merely reducing the level of detail in the descriptions of domain functionality. Ascertaining the right (or at least a useful) explanation for the variability in the domain required an ability to elicit a great deal of contextual information.

Approach

The ODM approach to this refinement process makes heavy use of the data that has been generated so far in the domain engineering life cycle. We have referred at various points in this document to the notion of the domain model defining a coherent "feature space" or "space of variations." To explore the interaction of any pair of features, one can take this idea literally by graphing their respective range of variability as the axes of a two-dimensional space. The *Describe Domain* sub-phase helped us identify the features and determine which cells are "filled in" with exemplars (legacy systems, known requirements for new systems). One goal of the

Exhibit 71. Refine Domain Model IDEF$_0$ Diagram
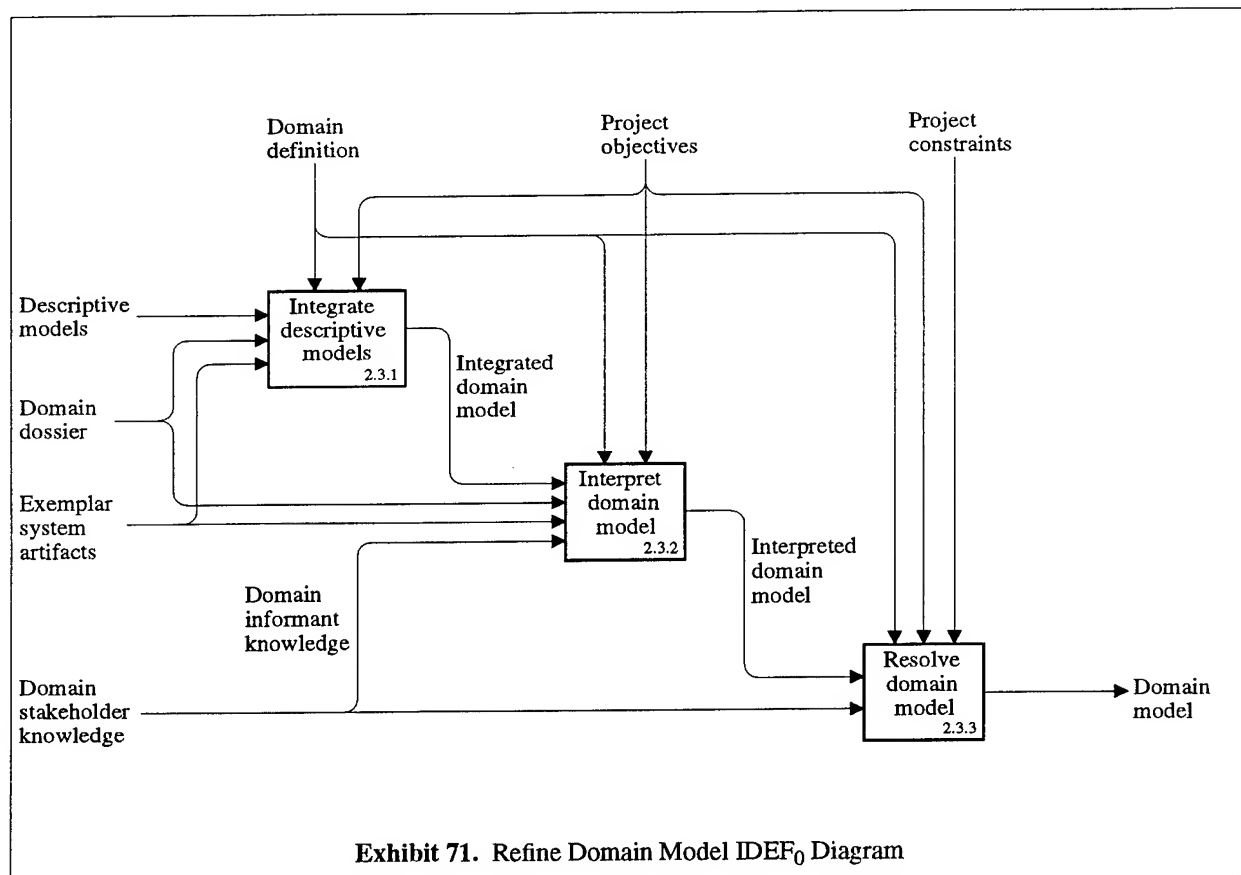
Refine Domain Model sub-phase is to help us systematically explore the gaps in the domain space, and, as necessary, restructure the space so that all elements are meaningful choices.

The name of this sub-phase, *Refine Domain Model*, emphasizes that we now begin to work increasingly with formal models directly. A general principle of the process in this sub-phase is to perform certain operations (e.g., integration of separate models, application of innovative transformations) on the models, and then validate the resulting formal results against empirical data (available from earlier work or newly acquired as necessary). We also do some more formal modeling of the settings in which domain functionality is embedded by identifying *setting characteristics*. Correlations are made that support an interpretation of commonality and variability in the domain in terms of relevant characteristics of settings in which featured capabilities occur, and rationale for why particular features (and *feature combinations*) are associated with particular setting characteristics. This interpretation allows for discovery of new feature variants, the discovery of innovative feature combinations, and the hypothesizing of new types of settings where the extended features could be of interest.

This approach to domain model refinement provides three important benefits:

- A basis for proper interpretation of domain variability;

- Controlled and systematic approach to innovation;

- Validating and stabilizing final boundaries of the DOMAIN MODEL that reflect but are not overly biased by empirical data collected in the *Describe Domain* sub-phase.

Results

The final output of this sub-phase is the DOMAIN MODEL. This model unifies the various concept and feature views of the domain into an integrated framework in which each element of the model is correlated with specific exemplar systems.

Process

As shown in Exhibit 71, the *Refine Domain Model* sub-phase has three tasks:

- In the first task, *Integrate Descriptive Models* relationships and interconnections between the separately developed DESCRIPTIVE MODELS are documented and formalized in the INTE-GRATED DOMAIN MODEL. This workproduct can be considered the ***domain-specific meta-model*** or "model of models" for the domain.

- The *Interpret Domain Model* task enhances the INTEGRATED DOMAIN MODEL with contextual information and rationale to explain why the representative systems differ in the observed ways.

- The *Resolve Domain Model* task uses specific model transformation techniques to extend the purely descriptive model to a more coherent and orthogonal *feature space*, and uses the contextual information from the *Interpret Domain Model* task to project potential usage settings for various innovative *feature combinations*

The DOMAIN MODEL must be *at least* comprehensive enough to cover the variability across the REPRESENTATIVE SYSTEMS SELECTION. But exemplars for the final DOMAIN MODEL need not be confined to the REPRESENTATIVE SYSTEMS SELECTION. Any system from the EXEMPLAR SYSTEMS SELECTION can be linked to the model. In the final task, where *innovative features* may be explored, even hypothetical exemplars can be introduced if necessary. Thus, while the linkage to empirical examples remains strong throughout the process, there is a gradual relaxation of the strictly descriptive nature of the modeling process over the course of this sub-phase.

There are a natural rhythm to this task sequence. Integration naturally precedes interpretation, as it leads to comparison of separately modeled aspects of the domain that create insights about hidden contextual information and rationale. We often do not ask why a system has a given feature until we have seen an example of a system with a variant feature.

The distinction between the tasks can be seen more clearly in detailing the gradual refinement of feature binding time information throughout the sub-phase. In the *Model Features* task within the *Describe Domain* sub-phase, modelers have associated specific features with specific settings (e.g., development or usage setting). During the *Refine Domain* sub-phase this picture is iteratively refined:

- During the *Integrate Descriptive Models* task, separately developed FEATURE MODELS are linked into a single INTEGRATED DOMAIN MODEL. This yields the more detailed comparative data needed to link features with specific feature binding sites within each setting.

- During *Interpret Domain Model*, links are established between feature variants that represent the same *logical* feature associated with different binding sites (possibly in different settings) across systems. Establishing these links requires both the INTEGRATED DOMAIN MODEL and additional DOMAIN INFORMANT KNOWLEDGE that allows correct interpretation of feature semantics, since the litmus test for this logical association of features is the viewpoint of domain practitioners.

251

- Finally, in *Resolve Domain Model*, new binding sites for precedented features can be explored by shifting features to new domain settings (e.g., compile-time to run-time).

Later, during the *Scope Asset Base* sub-phase of *Engineer Asset Base*, both precedented feature variants and potential innovations will be evaluated for utility and feasibility, and a subset of these will be selected for the intended scope of the asset base to be developed.

> *Example.* In the Outliner domain, the feature "ability to assign a font to a given heading level" can be realized by operations performed at a number of different feature binding sites (e.g., by selecting a heading individually and changing the font, by setting a style for the entire outline, by having a system session default, by assigning the heading a style which can itself be reassigned, etc.) In the case of a reassignable heading style, whether a change to a style affects headings already defined with the style or not is a more fine-grained binding-time issue. Modeling is required to equate the preferences file in one system with a differently named, but functionally equivalent "style format" in another system.

Guidelines

- The *Integrate Descriptive Models* task should be complete before beginning the *Interpret Domain Model* and *Resolve Domain Model* tasks. The *Interpret Domain Model* and *Resolve Domain Model* tasks each have the potential for an explosion of scope without the grounding of the fully INTEGRATED DOMAIN MODEL as a starting point. Discoveries of new data in the first task will naturally create tension to move quickly forward to interpretation to explain the anomaly. Modelers can choose to follow a refinement thread forward in this way, or, in a more breadth-first style, to complete a full pass of integration activity before moving on to interpretation.

- Some iteration between the *Interpret Domain Model* and *Resolve Domain Model* tasks is natural. Innovative ideas for features need to be validated by reinterpreting existing data; conversely, eliciting explanatory rationale from informants may require some hypothesizing of alternatives. The tasks can also be done in parallel; however, managing this concurrency will require extra discipline on the part of the modeling team.

6.3.1 Integrate Descriptive Models

The *Describe Domain* sub-phase has produced a DOMAIN LEXICON and a set of DESCRIPTIVE MODELS for the domain. As a first step in refining these models into the overall DOMAIN MODEL, the goal of this task is to produce a single INTEGRATED DOMAIN MODEL.
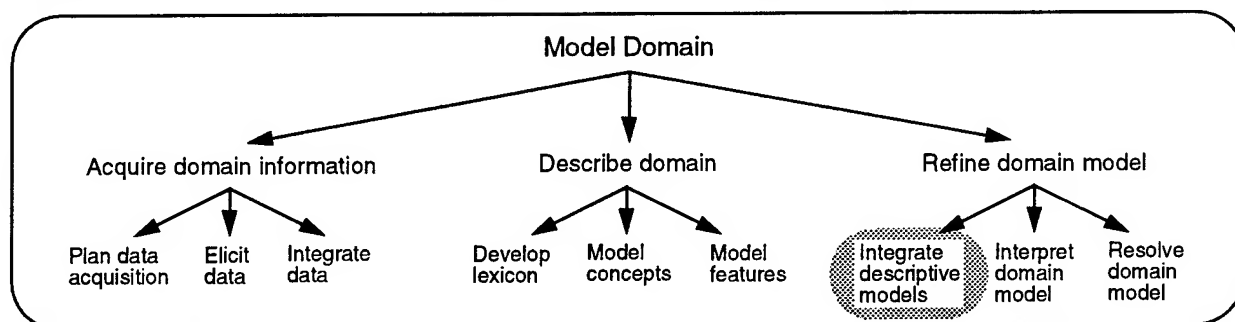
Exhibit 72. Integrate Descriptive Models Process Tree

- *Validate Descriptive Models.* This validation activity takes place at the beginning of the refinement stage because integration issues that are uncovered at this stage generally will

require interpretation to be resolved, and may thereafter lead to useful extensions of the domain model. In this sense the entire refinement process can be seen as having a validation function. While analogous to the notion of unit and integration testing in building a single system, the analogy breaks down because a domain asset base is not built in response to a single set of requirements. The separate viewpoints, alternative orders of seeing and comparing data are important for robust cognitive results of the process.

- *Allow for multiple perspectives.* The domain modeling phase represents a process that can be implemented with a variety of strategies leading to separate descriptive models. While ODM does not prescribe a particular strategy, the method does acknowledge that in practical settings, some divisions of this kind will be necessary. Building separate models, then integrating them into a unified model, seems to be an essential dynamic of the overall process.

- *Provide a single model for future iteration.* After completion of domain modeling, subsequent evolution steps can iterate back to the INTEGRATED DOMAIN MODEL, rather than back to individual FEATURE MODELS. These thus take on more of the nature of transitional workproduct, which reflect artifacts of team strategy, representation limitations, the order in which data was examined, modeler bias, etc. Evolutionary extensions may take the form of new data, new individual feature variants, or entire new ranges of features.

Approach

The ODM approach to *Integrate Descriptive Models* is based on the idea that many small models will be developed that codify diverse viewpoints into the domain. Integration involves some actual merging or fusion of models with similar viewpoints into a single model.

For example, if two different modelers created fragments of models dealing with end-user operations within the domain, integration may involve removing redundancies, resolving inconsistencies, filling gaps, etc. resulting in a single model. Unlike the move from unit to integration testing in system development, models cannot be partitioned in advance with clean interfaces in a manner analogous to top-down design of software system modules. So this integration work is likely to be a bit more messy, involving some semantic negotiation. One good rule of thumb is to work for a common model that preserves all the semantic distinctions provided in the independent models, rather than trying to reconcile them away.

Model Relations

Another key integration task involves creating an overall *map* of the models developed so far. Rather than trying to combine all the models into one monolithic domain model, the goal here is more of a *federation* of models. This requires, however, some approach to *meta-modeling* or building a model that can reference other models as elements. Since these other models might be quite heterogenous this is a challenging formal task.

When we model several concepts, there can be relationships between the things referred to (compilers translate code, outliners present outlines, tables of contents summarize books, etc.) These relationships are codified by the modeling formalism. Once concepts are placed into different models, there are also a set of relationships between these models. The types of relationships that can usefully be identified between or among models can also be deemed a formalism; however, one more constrained than the formalisms chosen for individual models.

We have not indicated a call to a supporting method for a formalism to support model relationships, as this is still an area of active research for domain modeling. Here, we suggest some types of model relations that we believe will be useful to consider in domain engineering, whether or

not they are supported by a formal representation or automated tools. Note that while specific models an specific relationships between models will depend on the domain, the types of relationships that occur between models should transfer across domains.

- *Features*: Concepts in one model (F) can serve as features for concepts in another model (M). This means that the concepts modeled in F can be correlated in some consistent way with features that differentiate the concepts in M. Relationships among the features are expressed (indirectly) in F, while relationships among the things modeled by M are expressed in M.

  Example. In the Chair domain introduced in an earlier philosophical moment, the concept Color served as a feature for the Chair concept.

- *Coextension*: This means that the things modeled by M are the same things modeled by N. Various gradation of this might be interesting, for models whose extents overlap in various ways. This relationship allows us to have concept models that are, in effect, alternative views of the same exemplar set. This gets us out of certain problems in ODM, for example, what to do with informant data that comes to us already modeled via, say, taxonomies.

- *Meta-model*: If model M is one of the things modeled by model N, then N is a meta-model for M.

There are many other potential model relations that can be useful to capture. Starter models may be included in the overall *model web* that represent preexisting models adapted to the domain. One model can also *specialize* another. Specialization captures the relationship that might hold between, say, an ODM standard top-level taxonomy for some area like "information sources" and project- and domain-specific extensions to this model. Legacy models are domain models inherited from previous modeling efforts or otherwise created by domain practitioners.

Workproducts

■ INTEGRATED DOMAIN MODEL

A workproduct which contains a unified and integrated model of feature variants synthesized from the individual FEATURE MODELS. This model includes:

- *Feature binding sites*. Integrates the FEATURE MODELS with DOMAIN SETTINGS. Each feature is associated with a specific site in a domain setting. In addition, domain practitioners who play various roles in those settings have various interests in and visibility to the features. In this task these relationships are refined to a next level of granularity, to individual feature binding sites within each domain setting and their relationships to practitioner roles.

- *Integrated descriptive model*. A linked version of the separate CONCEPT MODELS and FEATURE MODELS with redundancies and inconsistencies removed.

- *Representative systems feature profile*. A classification of each representative system in the REPRESENTATIVE SYSTEMS SELECTION in terms of features in the integrated descriptive model.

- *Domain model interconnections*. A **meta-model** that shows relationships between the descriptive models developed.

When to Start

You are ready to *Integrate Descriptive Models* when:

- At least one representative system has been studied in detail. Part of this task involves verifying the adequacy of features in the integrated model to fully characterize systems in terms of domain features.

- A set of individual FEATURE MODELS have been completed and all required validation for individual models has been performed.

  Some early integration activities can be done to validate the planned integration mechanisms — in theory, as early as the first definition of CONCEPTS OF INTEREST in the *Describe Domain* sub-phase. But the major part of the task should be performed when all individual FEATURE MODELS are complete, not as a pipeline of updates to the DESCRIPTIVE MODELS. Otherwise a lot of repeated integration effort will be expended.

Inputs

- DOMAIN DOSSIER. Information and characterization of the REPRESENTATIVE SYSTEMS SELECTION, used to derive the feature profiles for each representative that validates the INTEGRATED DOMAIN MODEL.

- DESCRIPTIVE MODELS. The separate models to be integrated. These include the DOMAIN LEXICON, CONCEPT MODELS and FEATURE MODELS which are to be merged into one INTEGRATED DOMAIN MODEL.

- EXEMPLAR SYSTEM ARTIFACTS. For validating the model against an exemplar outside the representative set.

Controls

- PROJECT OBJECTIVES. Indicates the objectives and audience for the final DOMAIN MODEL, which helps determine the completeness of the DESCRIPTIVE MODELS.

- DOMAIN DEFINITION. The DOMAIN DEFINING RULES, DOMAIN INTERACTIONS and DOMAIN CLASSIFICATION are used as references to maintain appropriate boundaries and interfaces for the INTEGRATED DOMAIN MODEL.

Activities

The overall structure of the activities in this task involve validating the internal consistency of the descriptive models, formally integrating them, validating the resulting integrated feature profiles against representative systems; finally, when these steps are done, addressing the question of how to build a true INTEGRATED DOMAIN MODEL given the available representational power of the models used. The following paragraphs describe the activities in more detail.

➤ Validate lexicon against DESCRIPTIVE MODELS

The greater degree of semantic analysis undertaken in concept and feature modeling creates opportunities to validate and verify initial decisions made in the *Develop Lexicon* task. One typical case is that two or more domain-distinct concepts have been clustered as one canonic term in the DOMAIN LEXICON.

Scenario: Insufficient resolution in domain terms

A canonic term has become a concept in a CONCEPT MODEL. Features (reflecting stakeholder

255

interests) are defined that differentiate that concept model. When the features "beam their way" down to the offending concept, we discover that the feature is more descriptive than the concept (in the context of its model). We may discover this by linking a set of exemplars to the concept and finding that no one feature value applies to all of them; or by finding feature linkages that appear to be mutually incompatible associated with the same concept.

Repair strategy: Go back to the DOMAIN LEXICON to find the synonyms for the canonic term. Go back to the DOMAIN DOSSIER for each synonym to discover whether the feature of interest maps different values to the different synonyms.

—   If this is the case, we have discovered, essentially, that a so-called synonym relation has masked a subtle shading of meaning significant for the domain. Therefore we unbundle or split the canonic term in the DOMAIN LEXICON, introducing new canonic terms for the needed synonyms. These new canonic terms are then introduced as new concepts into the appropriate CONCEPT MODEL, suitably restricted by feature values.

—   If the feature of interest does not seem to fit the synonyms, there are a few choices:

    —   look elsewhere in the DOMAIN LEXICON or DOMAIN DOSSIER for terms that seem to match;

    —   invent new terms and flow them back into the DOMAIN LEXICON; or

    —   consider whether the feature of interest is dealing with a domain interface feature. In this case you can add the feature of interest to the features associated with a related domain in the DOMAIN INTERACTIONS.

When you find the need for a new concept, always check the domain for relevant language first before imposing or inventing our own.

Scenario: Domain-irrelevant distinctions in terms.

In this case, the converse of the one above, two distinct canonic terms in the DOMAIN LEXICON are discovered not to yield domain-relevant distinctions. This is a problem because it adds distracting distinctions into the overall modeling language for the domain.

Repair Strategies:

—   Re-gather the terms under one canonic term; or

—   Expand the FEATURES OF INTEREST to catch the distinction between the terms.

➤ Integrate multiple concept views

Several CONCEPT MODELS may have been developed describing different practitioner views of the same representative systems. These could include legacy models as opposed to models created by the modeling team, or overlapping models done by different members of modeling team. At this stage it is desirable to merge these models. The merged models are documented in the integrated feature model component of the INTEGRATED DOMAIN MODEL.

The exact form of integration will depend on the Concept Modeling supporting method(s) chosen for the individual models and the overall integration plan. A shallow integration is possible merely by representing both models as alternate views in the domain model interconnections mapped in the INTEGRATED DOMAIN MODEL. A more challenging solution is to create a model that represents a literal *merging* of the alternative viewpoints of multiple models. The principle

here is that the merged model should be at least as expressive as the models it merges. That is, it should preserve all semantic distinctions made by each constituent model (forming, as it were, the "least common multiple" and not the "greatest common denominator" of the models).

➤ Integrate multiple feature views

In the *Model Features* task different sets of features may have been defined for the same initial CONCEPT MODEL. Cross-check these features for redundancy and overlap. Adjust model boundaries where necessary. The merged models are documented in the integrated feature model component of the INTEGRATED DOMAIN MODEL.

> *Example.* Exhibit 73 shows an example of two feature models to be integrated in the outliner domain. Model A describes some of the variations in how fonts (or more generally, display-specific attributes) can be associated with a heading within an outline structure.

> Model B is a simple model of the options for selecting the number of heading levels supported by a given outliner application. Most of the available exemplars have a hard-coded maximum number of levels (the "fixed at program compile time" variant) although the number of levels span a considerable range. A variant that is hypothetical with respect to the available legacy systems, but does represent a data point in terms of the known requirements for outliner applications, would be to have a fixed maximum but the ability to restrict this maximum to save memory for a given application. One could imagine this being implemented in various ways, including a compile-time or link-time constant, or a run-time variable that controls the memory allocation requests.

> Suppose these two model fragments had been created by two different modelers in the course of the effort. As part of the *Integrate Descriptive Models* task, we now need to consider the
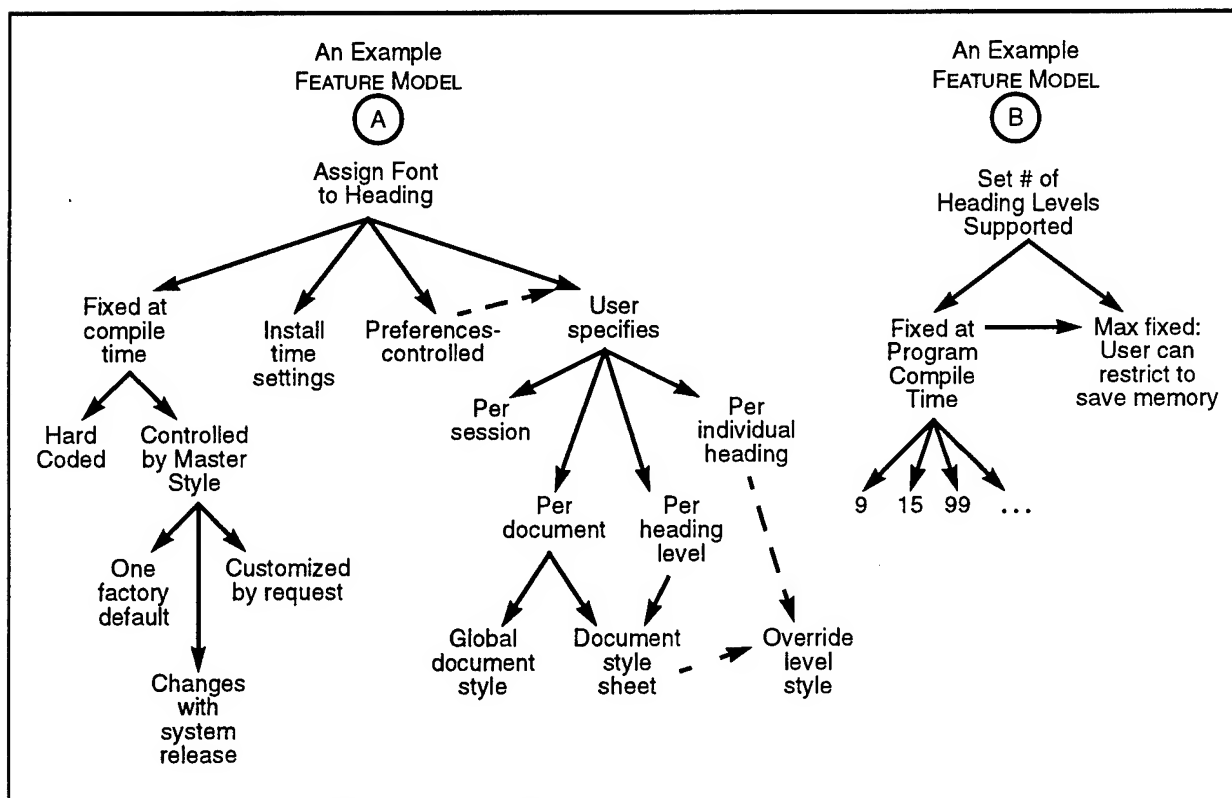
Exhibit 73. Examples of Feature Models to be Integrated

possible overlaps or interaction of these features. In this case, by studying the mapping back to exemplars we find that systems with hard-coded font attributes for levels also tend to support limited numbers of levels. For example, a text editor that supports nine levels provides hard-coded attributes for those levels. In contrast, systems that provide "up to 99 levels" of heading structure are more likely to provide direct support for specifying the format of each level via some kind of style sheet. Observation of these correlations is part of the integration task. Because these two models use a taxonomic form, integration by creating a single composite model is possible by adding some links between the models (not shown).

➤ Correlate features by setting

In the *Model Features* task of the *Describe Domain* sub-phase, features were linked to specific domain settings. In this activity, gather into a unified model all features defined for given entities (artifacts, processes, people) within a given setting. Compare different feature variants occurring in the separate FEATURE MODELS associated with this domain setting. Eliminate redundancies, resolve naming inconsistencies and identify apparent gaps in the resulting feature profile as required. Establish the required linkage so that a given artifact, or a given system in the domain, could be characterized by a single, minimally overlapping set of features.

➤ Identify feature binding sites

For each domain setting, identify the set of distinct feature binding sites: points within the setting at which feature variants may be selected. Identify feature binding sites by the following procedure:

1) Cluster variants associated with the same practitioner roles (e.g., requirements analyst, designer, end-user); and/or compare practitioners' work scenarios and look for common decision points for different feature variants (e.g., several variants are determined when a given configuration file is edited).

2) Distinguish different sites if the activity or mechanism differs, if the decision points happen at distinct times, or if different practitioners have responsibility. This is the rationale for using the more general term "binding site" rather than the more conventional "binding time" which draws a clearer analogy to variable binding time in programming. In the domain engineering context, there is not always a single temporal line along which various sites can be positioned. The notion of binding site encompasses the elements of time in workflow, time in program execution, and different practitioners in different settings.

3) Document the binding sites identified for each setting, together with the associated practitioner roles, in the INTEGRATED DOMAIN MODEL. Use names for binding sites that are familiar to domain practitioners. Some names may have been previously included in the DOMAIN LEXICON. If so, the FEATURE MODELS may already be structured in terms of the feature binding sites.

4) In the final formal step, reorganize individual FEATURE MODELS so that variants are differentiated where appropriate by reference to the associated feature binding site. It requires non-trivial analysis to link various system features together that represent the same "logical feature" in different binding site configurations, in different systems. It may require associating an operation in one exemplar system with a configurable data item in another system, and with a designed-in system parameter in another. The common link (feature) in this case is where the decision is made as to the font in which a given heading at a given level will be displayed.

Example. In the Outliner domain: in the simple example shown in Exhibit 73, each sepa-

rately developed feature model embeds features that span compile-time and run-time feature binding sites. In the case of Model A, consider a system that has a hard-coded set of formats (font settings, etc.) associated with the various heading levels within outline structures. In such a system, there would be no operation provided to the application user to set these display attributes.

As the binding time moves towards user configurability, specific operations must be provided to set the attributes for a given level. However, even these operations may be indirect. For example, style sheets might be provided that include attribute settings for all levels in a single specification. Different style sheets can be selected for a given outline document, but there may or may not be a mechanism for a local override of the style sheet format for a single heading at a given level.

➤ Characterize representatives with integrated models

Using the completed INTEGRATED DOMAIN MODEL and the information from the DOMAIN DOSSIER, characterize a reasonable subset of the REPRESENTATIVE SYSTEMS SELECTION using the integrated sets of features. Characterize the systems in terms of individual functional features, global, system-wide features and contextual information. Record results in feature profiles for each representative system.

In the *Model Features* task, features were validated against some representatives. In that task, the primary objective was to validate that every feature had some precedent. In this activity, we want the validation to continue to a further point. Here we validate the full set of features we have integrated. The idea is that this set of features should provide an adequate feature profiling mechanism for describing resources in the domain.

The definition of adequacy will depend on the intended audience and uses of the model. The domain-specific PROJECT OBJECTIVES should provide input to determine the level of detail and viewpoint needed for the feature profiles. If the model is to assist people in gaining access to existing, legacy artifacts, then the amount of characterizing of artifacts via features may be modest. If, on the other hand, this characterization is intended to produce enough information to guide a reengineering of artifact into asset, a much finer degree of detail may be required.

> *Example.* Now that we have the model of collapse/expand versus hide/reveal, we go back to our representative set and model MS Word in terms of these features. We find out, somewhat to our surprise, that MS Word, which had appeared to be a full-featured outliner, does not have the ability to hide and reveal. At the time we acquired data for the system, the lack of this function was actually invisible; it has now been revealed through cross-checking the full feature profile against each representative.

This is an example of the situation where characterization of the representatives creates surprising results. Our immediate reaction will be to proceed immediately to interpretation: why does the system behave in this unexpected way? This is the step forward to the *Interpret Domain Model* task. At this point, it is at least useful to capture the question for future action. We then make a tactical choice between following a refinement thread from *Integrate Descriptive Models* into *Interpret Domain Model* (and possibly on to the *Resolve Domain Model* task), or staying more breadth-first. Continuing integration activities will yield more data and more questions. The pattern of the questions in the aggregate will often reveal some explanations and reduce the amount of new data that needs to be acquired for interpretation.

➤ Adjust models for gaps in data

You will inevitably find gaps in the data. If you go back to fill those gaps simply to complete the representative systems feature profiles, this constitutes iteration or a "loop back" to the *Acquire Domain Information* sub-phase. In this case you will need to update the DOMAIN DOSSIER but the DESCRIPTIVE MODELS should not change. That is, the concept or feature was presumably already precedented by some other exemplar.

The following types of gaps can occur:

- You may find some combinations which the model is not expressive enough to describe; this can be handled in several ways:

  — Iteratively adjust the model so that it covers the cases;

  — Even this late in the game, revisit and downscope the DATA ACQUISITION PLAN by excluding the troublesome representative system from the final scope of the model. This effectively changes the scope, not of the overall domain, but of the portion to be modeled. The later in the process this happens, the more careful you want to be.

- You may also find some unfilled model areas (no representatives). Again, this can be handled in two ways.

  — Even this late in the game, you can go looking for additional representatives to fill the missing links; or

  — You can tighten up the model so that these cases are excluded.

These four options — loosen/tighten model semantics, and expand/contract representative set — are the central, interdependent tradeoffs that must be negotiated as final *Integrate Descriptive Models* activities.

➤ Integrate contextual data

To the extent that contextual information has been modeled in the CONCEPT MODELS, integrate these models with the features. This activity will be completed more comprehensively within the subsequent *Interpret Domain Model* task.

➤ Formalize model relationships

The CONCEPT MODELS and FEATURE MODELS that have been developed are integrated into the INTEGRATED DOMAIN MODEL, a ***meta-model*** within which each concept or feature model is represented as an instance. Here is where the various model relationships described in the Approach section of this task should be documented and validated.

Some modeling representation and method needs to be selected in which to formalize these relationships between models. This can involve making use of a Concept Modeling supporting method; however, in general very few of such representations or methods will provide adequate support for this activity. It is not necessary to have a sophisticated formal meta-modeling capability to continue on with the *Refine Domain Model* sub-phase. The greater the level of support from the underlying formalism, the more aid will be provided in later stages of modeling, and in handling large or complex models.

While ODM does not entirely constrain the choice of this representation, there are several points that should be considered:

- The representation should have the capability of describing other models directly as entities in the model. To this extent, it is an instance of a *meta-model*, a model containing entities referring to other models.

- The INTEGRATED DOMAIN MODEL will be linked with the individual DESCRIPTIVE MODELS. This suggests use of a common method and representations for the DESCRIPTIVE MODELS and the model of domain model interconnections itself.

- In addition, the INTEGRATED DOMAIN MODEL may be refined by including linkage to *starter models*. This further restricts the supporting methods that are most suitable.

- Even if the formalism used cannot support them, the types of model relations discussed in the Approach section are important to keep track of and document, even if informally.

Significant work is being done on evolving supporting representations and formalisms appropriate for support of domain model integration and refinement. Discussion of this work is beyond the scope of this document.

When To Stop

The *Integrate Descriptive Models* task is complete when:

- All workproducts of the *Define Domain* sub-phase of *Plan Domain* are complete. These have been integrated with the DESCRIPTIVE MODELS and conflicts have been resolved.

- All workproducts of the *Describe Domain* sub-phase of *Model Domain* have been integrated and conflicts resolved. These include the DOMAIN LEXICON as well as the DESCRIPTIVE MODELS.

- The INTEGRATED DOMAIN MODEL provides an adequate basis for answering the question: "What specific features, with respect to the domain of focus, differentiate the REPRESENTATIVE SYSTEMS SELECTION?"

  Is the INTEGRATED DOMAIN MODEL an adequate basis for distinguishing representative systems as a whole? Does it constitute a rich enough language for distinguishing profiles of other systems in the domain? Have some important features been left out? Are there features included that are not essential to the domain of focus?

Guidelines

- <u>Single vs. multiple models</u>. An overall tradeoff to be considered is whether to try to unify the various model views into a single, monolithic model or to retain the small, modular models as distinct structures. One advantage of the single model strategy is that more consolidation and removal of overlaps takes place during integration. A single model is also easier to maintain with respect to configuration management. The advantages of multiple models include their relative ease of comprehension and the retention of more information about the derivation of each model. The modeling formalism selected also will have an impact on this decision. Scalability to large, complex models may be needed to support a single, integrated domain model. Meta-model capabilities may be needed to support an integrated view of multiple (possibly heterogenous) models.

- <u>Converge on model stylistic conventions.</u> In viewing the models as a whole, you will typically find that similar kinds of modeling situations have been encountered and handled in different ways. A key integration activity involves finding these discrepancies, determining where possible if a standard approach can be taken; documenting the resulting decision; then making a pass through all the models normalizing them according to the established conventions. Allow for the following range of possibilities:

  — There is a recommended/preferred way to do it.

  — There is no real difference between the approaches. But it is convenient to have a standard way (e.g., to allow automated tools to make certain strong assumptions about what will be found.) An example is whether comment blocks should precede or follow modeling statements.

  — There are contingencies that allow for one of a few variant approaches to be used. These can be documented and applied uniformly.

  — The convention must be determined on a case-by-case basis.

No matter how thoroughly these conventions may be been discussed at the start of the *Model Domain* phase, check them out again. You may find that agreed-upon conventions were not followed, were misunderstood, or did not apply as uniformly as expected. Also, new patterns and conventions will emerge for each project. This care over conventions is more of an issue on a modeling project than for a standard software engineering project, because domain modeling is a less mature discipline, and large-scale modeling tasks do not decompose as neatly as systems.

- <u>Validate with other exemplars.</u> Characterize a non-representative from the EXEMPLAR SYSTEMS SELECTION using the INTEGRATED DOMAIN MODEL. This is the jumping-in point for evolving the domain model on the basis of incremental feedback.

6.3.2 Interpret Domain Model

In the *Integrate Descriptive Models* task, we created feature profiles for particular domain settings, then generated profiles for a sampling of representative settings. This provides us with data for interpretation. Some rationale for the features of particular artifacts or representative systems may have been documented in the *Acquire Domain Information* sub-phase, but before the model of features was complete there was no systematic way of deriving rationale for the variability.
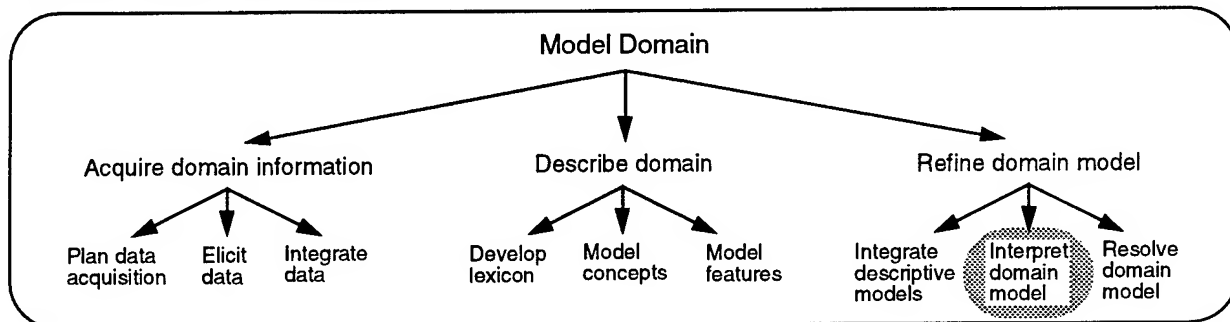
Exhibit 74. Interpret Domain Model Process Tree

The primary purpose of *Interpret Domain Model* is to establish contextual rationale for *why* particular systems have particular features, and why the features differ across the representative set. The INTEGRATED DOMAIN MODEL for the domain represents the required range of variability for the representative set of systems.

In general, **context recovery** for legacy systems is difficult. Documentation of process history and rationale is almost always fragmentary; the original system developers are often not available as informants. There is a certain amount of software "archeology" involved. By studying *multiple* representative systems in the domain engineering context, we can re-create some missing information; gaps in data about one system may be filled by data from another system.

Approach

The interpretation process in the ODM context essentially involves finding correlations between observed features of representative systems and other characteristics of those systems. We call these other characteristics **setting characteristics**: attributes of domain settings that appear to have explanatory value with respect to features associated with that setting. Setting characteristics are the restricted set of things we need to know about a given setting in order to guide feature selection for that setting. Note that having an integrated model of features serves to "tether" us in place, as controls on what characteristics of settings themselves need to be modeled. If we began setting characterization earlier in the process, we would lack this feature-oriented focus to guide the level of detail in description of settings. This would generally lead to *deep* descriptions of environment, settings, etc. taking us beyond the scope of domain modeling.[7]

The nature of the correlation between features and settings will vary according to the type of setting. The most typical settings considered in this guidebook, usage and development settings, can be broadly associated with characteristics that impact the **utility** and *feasibility* of various features. In this context, **utility** implies a correlation due to the usage setting (i.e., the feature is of interest to system users, or is not of interest). **Feasibility** similarly implies a correlation in terms of the developer setting (the feature is easy or difficult to implement, given a particular system approach). Usage settings may be further differentiated according to broad classes of relative interests based on the kinds of work activities being performed; developer settings according to the primary design alternatives available for implementing domain functionality.

Correlations addressed in *Interpret Domain Model* are of three varieties:

- *Feature to feature correlations*: We look for **feature clusters**, patterns of features that consistently co-occur (or consistently *never* co-occur) across exemplars. These include features that co-occur in several systems, as well as **feature constraints**, combinations of features that seem to rarely co-occur. Some co-occurrences may have been observed in the *Integrate Descriptive Models* task, but those primarily concerned overlapping features. Here, in essence, we are looking for patterns of meaning in a large set of presumably independent variables. This is similar to a classic data analysis task.

- *Feature to setting correlations*: What are the characteristics of a given exemplar setting that influenced the particular configuration of features provided for that exemplar? Up until now, we have been cautious about attempting to model anything that was not within the domain scope, although supporting information has been collected in the DOMAIN DOSSIER. Now that we have a focused set of questions to answer, we are in a better position to selectively note characteristics of the setting that appear to have direct bearing on feature choices.

- *Setting to setting correlations*: We initially considered some of the historical relations between exemplar systems when we made our REPRESENTATIVE SYSTEMS SELECTION during the *Plan Data Acquisition* task. This helped ensure that we did not inadvertently pick a biased sample set to work from (e.g., all systems derived by ad hoc or leveraged reuse of a single

---

[7.] This technique is roughly analogous to analysis of related domains in the *Situate Domain* task of the *Plan Domain* phase. The selected domain helps set a focused window into the "domain neighborhood."

ancestor system). Because we are attempting to draw inferences from patterns observed across multiple systems, we need to understand historical relationships between the systems in more detail than previously required. Given that our representatives have ideally included some historically related and some relatively independent systems within the domain, we now need more thorough understanding of these relations in order to properly interpret the observed commonality and variability across the systems.

The basic *domain interpretation rule* could be summarized as follows:

> The larger the set of representative data, the more historically independent the representatives, the larger the cluster, and the larger the set of co-occurrences, the more significant the cluster is from the standpoint of domain evidence.

With these correlations to work from, we are in a good position to build a *theory* of the domain that predicts what feature combinations will be of interest in settings with particular characteristics. This theory-building will be completed in the next task, *Resolve Domain Model.*

There are several benefits of an explicit interpretation task. Within the *Model Domain* phase, the rationale helps provide a filter on various techniques used to formally extend the domain model into its final form (performed in the following process, *Resolve Domain Model*). Rationale for various features in the domain model will later be used in the *Engineer Asset Base* phase to identify potential customer interest in particular feature combinations.

Workproducts

■ INTERPRETED DOMAIN MODEL

An extension of the INTEGRATED DOMAIN MODEL adding:

- Feature constraints, which reduce the possible feature space to exclude combinations interpreted as out of scope or semantically meaningless.

- Feature clusters that appear to have strong rationale for co-occurrence patterns. These can be annotated as a list of clusters. For each cluster, document:

  — A provisional name/description for the cluster suggestive of the overall semantic interpretation of its significance;

  — The constituent features within the cluster;

  — The nature of the correlation between the features (i.e., strong co-occurrence or strong *absence* of co-occurrence, i.e., features that rarely or never are supported together);

  — Interpreted rationale for the cluster: Feature occurrences documented as historical *vestiges* or as strongly correlated with setting characteristics.

- Setting characteristics: attributes of domain practitioners or surrounding system settings that are relevant to the explanatory model of features derived in this task. These are correlated to the DOMAIN SETTINGS in the DOMAIN DEFINITION. The characteristics can be modeled formally similarly to features; i.e., each characteristic can be defined as a category that can take certain values. Exemplar settings that are instances of each domain setting have distinct values for each characteristic associated with that setting. The set of all values for an exemplar setting is its profile.

- Correlations between features or feature clusters and setting characteristics.

- Other documented process information, rationale, decision histories, trade-offs, and issues surrounding elements in the INTEGRATED DOMAIN MODEL.

When to Start

Individual interpretation activities will have been performed throughout the life cycle. However, the *Interpret Domain Model* task proper begins when:

- The INTEGRATED DOMAIN MODEL is complete. A comprehensive and accurate picture of the significant features in the domain provides the basis for eliciting relevant rationale and contextual information.

- Feature profiles are complete for some subset of the REPRESENTATIVE SYSTEMS SELECTION. These provide a basis for interpretation of variability in domain data. Interpretation may require elicitation of further information (e.g., analogous to further experiments to validate a scientific hypothesis) but should *not* involve iterations to verify initial data.

Inputs

- DOMAIN DOSSIER. Source of additional contextual information about the REPRESENTATIVE SYSTEMS SELECTION needed to answer the questions generated by examining feature patterns across the systems. Often the needed information will already have been gathered but will not have been interpreted in the appropriate setting.

- INTEGRATED DOMAIN MODEL. The primary input, which provides the features for interpretation. Also provides profiles of representative systems in terms of the integrated set of features, which are used as input to detecting and analyzing clusters (patterns of feature co-occurrence).

- EXEMPLAR SYSTEM ARTIFACTS. Required for new data acquisition activities undertaken to validate interpretations. Newly elicited knowledge about historical and genealogical relationships among the REPRESENTATION SYSTEMS SELECTION, needed to distinguish historical rationale for patterns of feature occurrences within settings.

- DOMAIN INFORMANT KNOWLEDGE. Required for new data acquisition activities undertaken to validate interpretations. Newly elicited knowledge about historical and genealogical relationships among the REPRESENTATION SYSTEMS SELECTION, needed to distinguish historical rationale for patterns of feature occurrences within settings.

  These inputs are considered distinct from the inputs required in the *Acquire Domain Information* sub-phase, because different kinds of information will be elicited and the results will be recorded, not directly in the DESCRIPTIVE MODELS, but in the associated information about domain setting characteristics and other contextual information. This is why the data gathering activities in *Interpret Domain Model* are not considered a simple "loop back" in the model.

  For example, it might typically be necessary to schedule a follow-up interview with a domain informant to ask more focused questions in support of model interpretation. It is not always necessary to do so, however. A skilled interviewer could move from pure descriptive data gathering to interpretive data gathering in the course of a single interview. Even in this case, however, the distinction between the activities would be important.

Controls

- PROJECT OBJECTIVES. Interpretation helps convert descriptive usage history into predictive, advisory rationale for selecting features in new application settings. Knowing the specific project objectives will help focus the interpretation activities on the needs of potential asset implementors and asset utilizers.

- DOMAIN DEFINITION. The DOMAIN CLASSIFICATION provides *analogy domain* relationships as one source for interpretation of *feature configurations*. The DOMAIN HISTORY provides genealogical information at the domain level which may also be significant.

Activities

The basic flow of activities in this task involves a correlation, analysis and hypothesis-building stage, a round of additional data collection to validate the interpretations, and synthesis of the conclusions into some broad categories within the model. The initial correlation process has a center-to-periphery movement, focusing first on feature-to-feature correlations, then moving to associated setting characteristics based on the feature correlations, and finally looking for possible historical relationships among the exemplar settings to cross-check the explanations of common and variant features. One advantage of this overall process design is that it helps keep a tight focus on the setting characteristics considered (which could otherwise become unmanageable, since the boundary conditions of the DOMAIN DEFINITION do not provide a direct filter), and on the new data considered.

One risk to consider throughout the task is the fact that the search for patterns and correlations, like other modeling activities, needs its own criteria for completeness and closure. There will always be interesting new patterns revealed with each round of analysis. The PROJECT OBJECTIVES are therefore an essential control on the overall activity. At every point, keep in mind that the correlations will eventually be used to help guide asset utilizers to assets with the right feature profile for their needs. Consider which explanations will prove of value to the final customers.

➤ Cluster features by co-occurrence

Identify combinations of features with significant patterns of co-occurrence. Working from the REPRESENTATIVE SYSTEM FEATURE PROFILE within the INTEGRATED DOMAIN MODEL, look for significant patterns of co-occurrence across systems. These feature clusters include combinations that co-occur in several systems, as well as combinations of features that seem to rarely co-occur.

Document these by annotating the REPRESENTATIVE SYSTEM FEATURE PROFILE, so that the features within each profile that have been "clustered" are noted. Note that the clusters can overlap, so that the same feature may show up as a member of multiple clusters. The clusters also need not cover the feature profile, so that there may be some features that are not included in any cluster.

Also begin creating the INTERPRETED DOMAIN MODEL by starting a list with entries for each cluster. For each cluster, annotate the following:

- Does the cluster represent a co-occurrence or a constraint (positive or negative correlation between the features)?

- Linkage to the representative systems that match the cluster hypothesis.

  Example. In the Outliner domain, not all outlining programs support the operation of "incremental expand" or "incremental collapse." This operation allows the user to keep a cursor positioned at a given structural item in the outline and expand the current structure by one

more level (relative to its current state). Though not all outliners support these operations, when they support incremental expand they almost always support incremental collapse as well.

As an example of features that rarely occur in tandem,

➤ Characterize settings

Once we have feature clusters, we can begin providing contextual information for *why* the features occur as they do. For each feature cluster, identify the setting characteristics that appear relevant to particular feature configurations. This means going back to rationale for individual systems that may have been modeled in the INTEGRATED DOMAIN MODEL or captured less formally in the DOMAIN DOSSIER. Even though a feature may be associated with a particular setting the rationale for its inclusion may rest in another setting. (I.e., the system has this feature because it was easy to implement, not because users needed it!)

The DOMAIN SETTINGS within the DOMAIN DEFINITION provide a set of categories for different kinds of settings in the system life cycle. Annotate the DOMAIN SETTINGS as you learn what characteristics of settings are relevant for particular feature choices.

➤ Map setting historical linkages

One important area to apply this analysis is in building correlations between features/feature clusters and setting characteristics where there appears to be weak motivation for the correlation. If features have been imported into a new system as a result of *ad hoc* reuse then there may not be a strong correlation in terms of utility. Consider historical relationships between representative systems, particularly in cases of apparent anomalies in the data. A primary heuristic to apply here is the following:

> The closer the historical or contextual closeness of two representative systems, the more interesting is the variability in their feature profiles. Conversely, for systems that are historically or contextually diverse, commonality in feature profiles is of particular interest.

Certain features begin to occur across systems because these systems are competing in a "feature-driven" marketplace. These historical and market patterns will tend to obscure the rationale for features that are the intended result of the *Interpret Domain Model* task. Use information from the DOMAIN HISTORY workproduct to identify systems that may represent earlier technology or requirements that reflect trends towards new technologies.

> *Example.* In the Outliner domain, outlining programs that evolved from an "idea-processor" perspective differ in fundamental ways from programs that treated outlining as an extension of document processing. While both categories are considered outliners, the set of operations supported and even the underlying data structures differ considerably.

➤ Form hypotheses and questions

Based on observed feature clusters and other patterns, form hypotheses about the rationale for the co-occurrence patterns. Use documented rationale from the DOMAIN DOSSIER as a starting point. Document the results in a list of hypotheses and questions for investigation that would validate the hypotheses.

➤ Elicit supporting data

Validation of the interpretive hypotheses will generally require additional data acquisition. Use the questions as basis for further elicitation, artifact analysis, and observation. Acquire information about features using the same techniques and supporting methods as were employed in the *Acquire Domain Information* sub-phase.

However, the process here is quite different. In the *Acquire Domain Information* sub-phase rationale was gathered for specific representative systems or artifacts. In the *Interpret Domain Model* task, the intent is to develop rationale explaining the commonality and variability across the REPRESENTATIVE SYSTEMS SELECTION. If you loop back to the *Acquire Domain Information* sub-phase to validate one of these interpretations, then finding a counter-example might result in retracting some assumptions built into the model or in refining the model by increasing its predictiveness and inferential capability. These means that modeling changes must also be made, so the iteration must include re-performing the *Describe Domain* sub-phase.

It is often difficult to recreate contextual information by working strictly from typical static sources (e.g., documents, code, etc.) Study of system users' practice may be helpful in revealing the rationale for certain features, or it may reveal that some features were hidden as part of the work practice in the usage setting, where functionality is performed through system workarounds, routine sequences of primitive system operations, informally reused data, etc.

This is a particularly good time to bring diverse informants into contact with each other as part of the validation process.

➤ Model constraints and dependencies

Constraints, in the context of the FEATURE MODELS, might be best thought of as "negative features." A constraint prevents you from creating certain model configurations based on rules. A dependency allows you to infer certain additional information based on rules. Although the distinction is supporting method-dependent, any supporting method and representation should distinguish these in some way.

Constraints are valuable in differentiating features that are intentionally rather than accidentally excluded from the domain range of variability, or cases where a supported feature in one setting is specifically required to be absent in another setting. Constraints can simplify other models, by selectively filtering excluded instances, thus allowing the taxonomic model to define a superset of the instances intended for coverage. They can help control the potential combinatorial explosion of variation in domain modeling.

> *Example.* In the Outliner domain, one system enforces constraints that disallow creation of outlines in "improper style" (i.e., a heading of level 3 immediately following a heading of level 1). Other programs allow such configurations. The difference is *not* merely one of greater versus lesser functionality. If the intended use of the tool is as a strict outline manager or idea processor, the more restrictive version of the tool may in fact be more useful. The less constrained variant is patterned more after the notion of styles in a document format, and is more typical in applications where outlining is embedded within broader text-editing capabilities.

The degree to which constraints and dependencies can be directly integrated with concept and feature models depends on the sophistication of the modeling representations used, the specific constraint mechanisms they support, and the presence of other representation capabilities such as multiple inheritance.

➤ <u>Correlate feature clusters and setting characteristics</u>

As the final activity in the *Interpret Domain Model* task, synthesize the interpretive information derived so far by identifying large-scale correspondences that emerge from the analysis between feature clusters and identified setting characteristics. This step marks the transition to the next task, *Resolve Domain Model*, where innovative combinations of features will be explored.

The output of this activity will generally by a higher-level clustering (clusters of clusters) of features, correlated with setting characteristics. The correlation can be further qualified according to a simple model: positively versus negatively correlated (clusters versus constraints) according to utility and feasibility.

> *Example.* In the Outliner domain, interpretation yields two broad categories of outliner application, exemplified by the text editor and the directory browser as contrasting cases. The dominant characteristic of the "text editor" mode is that the content being manipulated by the outliner is under the direct control of the application itself (e.g., an outline mode-browsable document). In the "directory browser" mode, the outliner is being used as a window onto a collection of relatively independent data.

> The implications of these two paradigms are manifold; they affect a number of different feature clusters and setting characteristics. For example, the text editor outliner can typically provide features like "incremental expand/collapse from a fixed cursor position." From an implementation standpoint it is the application that is retaining the necessary state information and executing the display commands to effect this. For the directory browser application, this would be much more complex to effect. Typically, display state information is associated with each folder or directory, and so a logically coherent single operation from the user's standpoint (unfold outline one more level) would require a behind-the-scenes updating of many separate data structures.

> There is also the question of utility of the feature. For viewing documents, incremental expand/collapse is useful because we like to visualize a document at different levels of detail. The same operation is less useful in a directory browsing environment, where we are more typically navigating through the structure searching for something in particular. These are not absolutes. They are explanatory models for the rationale of why certain implementation strategies are chosen and certain features provided.

> On the other hand, consider the "hide/reveal" feature; the ability to "windowshade" a portion of an outline closed, then re-open it with the same pattern of open and closed sub-headings remembered from before. From an implementation standpoint, this is much harder to achieve in the text editor outliner paradigm. Outliner *operations* are managed from the application, but there is no mechanism to retain local state information on a heading-by-heading basis. What appears a simple operation turns out to violate deep-rooted assumptions buried in the overall architecture of the application. For the directory browser application, on the other hand, this operation is easy to support, since open/closed state information is effectively distributed recursively throughout the separate hierarchical folders/directories being browsed. In this case, it is harder to argue that the feature would be of less interest or use in document editing as directory browsing; the explanation seems to be implementation-driven rather than needs-driven.

> As another, and more familiar example, the related domain of "interactive text editors" can be broadly partitioned into editors that support a moded versus a modeless style of operation. While in principle "modedness" itself could be considered a feature, its semantic repercussions, interactions and impact on other features is so pervasive as to warrant its representation as a strongly correlated set of feature variant choices.

When To Stop

You can complete the initial pass through *Interpret Domain Model* when:

- The INTERPRETED DOMAIN MODEL provides a rationale for observed commonality and variability in the REPRESENTATIVE SYSTEMS SELECTION. The model adequately answers the question "Why are the systems different in the observed respects?"

- A given *descriptive feature* or feature cluster can be mapped, using the model, to some setting characteristics that suggest circumstances when the particular feature(s) would be viable.

- Conversely, setting characteristics can be mapped to one or more features or feature clusters.

Guidelines

- Be alert for functionality withheld from end-users for definite and deliberate reasons. It can be as important to explain the absence as well as the presence of a feature in a given setting. Document such discoveries in updates to the INTERPRETED DOMAIN MODEL.

  Example. In some military applications where normally both read and write capabilities might be expected, write capabilities are intentionally factored out of implementations to avoid malicious changing of orders or for other security reasons. A system providing both read and write access would not be acceptable in this setting.

- Use analogy domain relations to elicit contextual rationale. When software developers implement systems, they both knowingly and unknowingly apply analogies that constrain the types of features they consider. If a feature cluster shows up in multiple settings, and there does not appear to be a strong need for all the features in the cluster, it may be an "importation" of features associated with some analogy domain active for the developer. Conversely, if there is a patter of consistent requests for feature enhancements from users that were not included in the original system, this may indicate a user-level analogy that is shaping expectations and the kinds of work practice surrounding the system. As part of interpretation, we could ask questions like, "What assumptions might the implementors have made because of this analogy about the ways that outlines would be used and the features desired? What existing features might be under-utilized? What new features might be expected by users that suit the analogy but are difficult to implement?"

  In the Outliner domain, one exemplar system used terminology drawn from the analogy domain of human genealogical relations with a preference for feminine terms (e.g., mother for a heading with sub-structure, daughter for a sub-heading, sister for headings at the same level and under the same "mother" heading, etc.) This exemplar provided a primitive operation to "Create Aunt," that is, create a new heading beneath and at one level higher than the current heading. Most other systems would require two operations to create an "aunt", create new heading (typically a sibling beneath the current heading), and promote the heading to one level higher in the structure.

  Since it equates to a composite of two operations in other exemplars, this is not a case of an operation that is fundamentally difficult to implement. But why was this operation implemented in the exemplar as a primitive? Was it based on analysis of the typical creation pattern for outlines (creating new lines downward, either siblings, daughters or aunts)? Supporting this hypothesis is the presence of a "create daughter" as well as "create sister" operation. Or was the analogy itself a factor (Did the presence of the relational term "aunt" in the analogy domain lexicon lead to discovery of a useful operation?) These questions and hypotheses could become the basis for a round of interpretive data

270

acquisition.

Example. An interesting example of the use of an analogy domain relationship to elicit hidden contextual information comes from the Army STARS Demonstration Project. Discussing the domain of Emitter Location Processing and Analysis (ELPA) in an information-gathering meeting, an analogy was made to car radio receivers. The analogy arose because, like drivers with their car tuners, ELPA operators can initiate operations to "seek" or "scan" various frequency ranges. By working backwards from the analogy, the question arose: did operators have functionality analogous to the pre-set station buttons on a car radio? The answer was no, and the rationale was "to prevent system operators from using the receivers as radios for their own entertainment."

6.3.3 Resolve Domain Model

As illustrated in the process tree shown in Exhibit 75, in the *Model Domain* phase as a whole, we have moved gradually from empirical examination of data in *Acquire Domain Information* sub-phase through descriptive, comparative modeling in the *Describe Domain* sub-phase. Now, having passed through the *Interpret Domain Model* task, we have at least a partially validated *theory* for the domain. We have workable explanations for why particular combinations of features occur (or do not occur) in given settings. This last step is essential in order to meet the objectives of constructing an ASSET BASE for the domain; without such explanations we have little basis for confidence that asset utilizers will find assets suitable to their needs, simply because of perceived similarities between their settings and those used as exemplars in descriptive modeling.
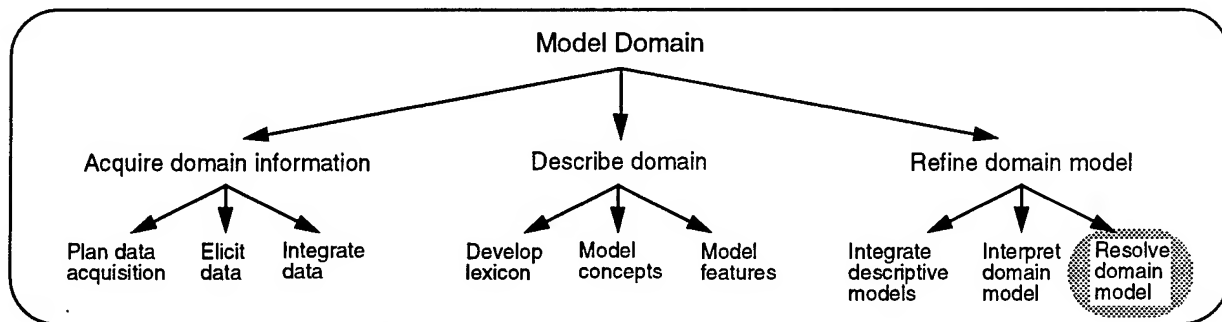
Exhibit 75. Resolve Domain Model Process Tree

The final task of the *Model Domain* phase, *Resolve Domain Model*, provides a bridge between the descriptive modeling and the prescriptive decisions made in the *Engineer Asset Base* phase. This bridge can be thought of as the "what *could* be" that lies between "what is" and "what will be." The goal is to extend the models developed to reflect a *coherent field of possibilities*. The result should be a DOMAIN MODEL with a high degree of conceptual cohesiveness, consistency, predictability and orthogonality.

There are several goals addressed by this task. First, the task serves as validation for the entire domain engineering process to this point. Specifically, it provides a chance to review and make final adjustments to the boundaries and domain relations documented in the DOMAIN DEFINITION on the basis of the knowledge gained in the *Model Domain* phase. Second, the task builds on the interpretation of domain information in order to simplify and organize the models. This may involve removing some special cases that make the model overly complex, as well as including some possibilities that extend the coverage of the model while simplifying its semantics. Finally, the task helps identify innovative features and potential new settings for applying domain functionality. Keep in mind that the objective of this task is not innovation for its own sake (although

271

in some environments such innovation may indeed be of high value). Rather, innovation techniques are employed to ensure certain closure properties on the DOMAIN MODEL.

These are challenging goals. To the extent that we set ourselves the task of exploring for gaps in the domain model, this is a theoretically unbounded task. Exploratory or learning-based processes are very hard to scope and manage. There is also a danger of expanding the boundaries of the domain in an uncontrolled way, or encountering a combinatorial explosion of feature combinations within the current boundaries. Once we depart from the iron rule of purely descriptive modeling (tracing all features back to exemplars such as legacy system capabilities or identified requirements for new systems) and consider possibilities, we have only the conceptual integrity of the domain to hold us into place. Yet if we do not venture out into this free space beyond purely descriptive features, the model is likely to remain somewhat brittle, easily perturbed by new exemplars as they are discovered.

Many aspects of the ODM process so far address the challenges of complex feature combinations described above. The emphasis on selecting and scoping relatively small domains, with manageable numbers of features but richly modeled semantic relationships, the care taken in defining domain boundaries and tracking changes to these boundaries, and the isolation of sub-domains as "black box" elements are all crucial to controlling *feature creep* in the DOMAIN MODEL. The *Interpret Domain Model* task has also helped identify constraints that can screen out uninteresting feature combinations. But additional techniques are needed to control the combinatorics of feature combinations and help derive robust and evolvable abstractions for the domain.

Approach

Predictiveness of the Domain Model

ODM treats workproducts built for a specific single usage setting as exemplars. There are inevitably gaps and overlaps in functionality across a collection of such exemplars covering a given domain (assuming they were developed in different settings for different applications). There are also traps, unknown dependencies, hidden features and contextual assumptions, etc. But these would be irrelevant if *feature sets* merely had to correspond to existing exemplars. The DOMAIN MODEL could in this case be a purely descriptive inventory.

However, the business of domain engineering is to help engineers figure out the best way to restructure and re-engineer a set of exemplar workproducts into a coordinated set of assets within an asset base. This means we want the DOMAIN MODEL to be *predictive*. When a new application context is considered, not considered as an exemplar during domain modeling, there should be a good chance that the features required for that setting are either included in the model, or are excluded by an identified boundary condition.

This does not mean that the model must satisfy a requested feature profile from any new application setting exactly. First, the overall application needs will almost always include extra-domain aspects not addressed by the model. Even within the domain, however, users may ask for specific features or combinations of features not supported by the model. They may also not request certain features the model suggests are essential.

> *Example.* In the Outliner Domain, suppose, after completing the DOMAIN MODEL for a set of representative outliner systems, that an asset base is implemented. We now consider a new set of applications, outlining for a mortgage broker application, where an outline interface is desired that will allow hide-and-reveal for mortgage data, broken down by brokerage firm, size of loans offered, regions, type of mortgage, etc.

In looking over the requirements for these applications, we find a number of specifics that are obviously concerned with the mortgage brokerage" domain and not the outliner domain at all. Are there outliner-specific functions required for this application that were not anticipated in the DOMAIN MODEL?

What we want at this point is for the DOMAIN MODEL to reveal interesting mismatches with requested features. The rationale is most clearly demonstrated first with an over-simplification. Assume that each feature set in the DOMAIN MODEL corresponds to an implemented asset that could be directly utilized if there were a close enough match to its feature profile. In this case, getting the customer (i.e., the application developer) to accept some extra, unneeded functionality or to forego some desired functionality would be part of the compromise needed to reuse the asset.

The guiding principle of the *Resolve Domain Model* task is that the DOMAIN MODEL can be *resolved* into a set of such feature sets, correlated with profiles of the characteristics of settings with which various feature sets would be associated. These feature sets are not directly tied to the feature clusters that correspond to legacy artifacts. They also need not represent a commitment to build assets for any particular feature sets. You can define more feature sets than those for which you intend to build satisfying assets. They represent a key set of abstractions that structure the commonality and variability within the domain.

When the feature set disagrees with a requested feature profile, the quality of the DOMAIN MODEL will be revealed in how much useful domain knowledge can be discerned in the tension between model and application. If an application calls for a subset of a specified feature set in the DOMAIN MODEL, a subset not supported as a separable entity in the model, our domain modeling experience says, in effect: "You are wrong. You don't want to take only those features, without the rest in this set as well." The other features in the set become, from the standpoint of the application, innovative features that might not otherwise have been considered. Conversely, suppose application requirements call for a superset of a specified feature set in the DOMAIN MODEL (i.e., the feature set plus some extraneous features.) Here the "duty" of the DOMAIN MODEL is to declare: "You do not want those extra features. They will not be stable and coherent. You should take this set only."

Note the distinction of this matching process from conventional requirements analysis, which does not include this notion of negotiating away requirements based on the cohesiveness of the requirements set, or talking the customer into adding new requirements (although this happens in practice all the time). But "design with reuse" or asset utilization is not the same process as conventional applications development, which is primarily problem-solving. Asset utilization is one component of a global optimization problem between an evolving set of needs and an evolving inventory of assets.

The feature sets can be used to guide "rough matches" to legacy artifacts, either for ad hoc/opportunistic reuse directly by application developers, or as input to asset implementors who will at least have the benefit of some adaptable prototypes and examples to work from. Even if application developers have to implement components satisfying the feature sets directly, these components will have a soundness with respect to the domain that would not have been obtained by mere responsiveness to the application requirements. These implementations in turn become strong candidates for conversion to reusable assets.

Mapping Feature Clusters into Feature Sets

In the *Interpret Domain Model* task, we discovered feature clusters that corresponded to empirically observed patterns of co-occurrence. Through interpretation we characterized these clusters as historical versus logical, accidental versus essential in nature. A primary objective of the

273

Resolve Domain Model task is to transform these feature clusters into *feature sets* that will have more durable predictive value for systems, or assets, implemented on the basis of the final DOMAIN MODEL.

What is the basis of this transformation? Like most refinement processes in ODM, it is not a simple selection or subsetting, nor a concatenation or aggregation of clusters.

- A cluster may be split, or certain features may be removed, because according to the interpretation the reason for the clustering was historical and non-essential. For example, if a certain cluster occurs consistently in systems that were all leveraged from an initial prototype system, the co-occurrence can be interpreted as resulting from this leveraged reuse. In this case (if there is other motivation for subsetting) a feature could be removed or a subset of features from the cluster could be isolated.

- Conversely, certain features that never co-occur in exemplar systems but appear to be compatible could be joined together in a new grouping (i.e., part of a feature set that did not appear within a cluster).

- Certain individual features can be eliminated altogether from the final DOMAIN MODEL in order to provide clearer boundary conditions, with fewer exceptions. This can be thought of as a kind of smoothing function where certain features represent outcroppings that are worth removing in order to have a more cohesive feature space overall.

- Conversely, it is possible to discover new features (not simply new combinations of existing features) that still fit within the domain scope but for which there was no precedent in exemplar systems.

These various transformation steps should make clear that the transition from feature clusters to feature sets is non-trivial and requires a great deal of analytical skill and judgment. All the knowledge gained in the domain is relevant to this process.

Innovative Modeling Techniques

The *Resolve Domain Model* task involves application of a repertoire of **innovative modeling** techniques. Once domain information has been transformed into a formal model representation, it is possible to apply transformations and operations directly on this model data in order to derive new domain possibilities. Such transformations can even be applied by those without extensive domain knowledge; in fact in some respects it may be easier for "domain-naive" modelers because they have fewer pre-conceptions about "what makes sense" in the domain. Of course, in the end domain practitioners must evaluate what is useful and feasible for the domain; but in the innovative modeling process it is helpful to defer these considerations. The techniques rely on the knowledge created by the earlier descriptive and interpretive processes. In particular, we are looking for innovations that we can get "almost for free":

- By viewing the overall functionality in the domain as a whole, it is often possible to see capabilities that do not require solving new problems, but merely re-packaging or re-configuring solutions that already exist.

- Sometimes, new capabilities arise in fact by *removing* functionality as well as adding it. It is often the extra work that a component performs that ties it to particular contextual sets of assumptions and prevents its ease of reuse in new environments.

- New capabilities can also be derived by placing existing capabilities in new settings. Innovation lies in the combination of features with settings, and new discoveries on either side of the equation can lead to important new value creation.

274

Taking the time to investigate these synergistic opportunities creates significant added value for the overall domain modeling process (as compared to, for example, a parallel set of system development efforts). This process provides an opportunity for innovative design and the discovery of novel opportunities, through combinations of features that have all been proved individually to be attainable. It can help reveal unexpected commonality across systems and even across domains.

This task can also lay the groundwork for a layered design approach that identifies core and peripheral sets of features for system versions within the domain. This layered approach may first be suggested in *Interpret Domain Model* if there are several distinct overall semantic interpretations for structures and operations in the domain. It also may carry over into the layered approaches in the *Architect Asset Base* sub-phase of the *Engineer Asset Base* phase.

Workproducts

■ DOMAIN MODEL

This represents the final form of the model produced in this phase. Besides the existing INTERPRETED DOMAIN MODEL this model includes the following:

- *Extended model of domain features.* Contains both precedented and unprecedented features and innovative extensions to features. The representation form used should be compatible with those used in earlier sub-phases. These features will be a superset of those in the INTERPRETIVE DOMAIN MODEL.

- *Extended setting characteristics.* Contains semantic information about attributes of potential settings of interest for model features, including hypothetical new settings.

The *Resolve Domain Model* task involves defining numerous connections between features and settings information. The exact partitioning strategy for the DOMAIN MODEL will depend on the supporting methods used in previous tasks.

When to Start

Ideas for innovations in domain functionality will occur throughout the modeling process. These can be captured informally as they arise (e.g., in a "things we would like to see" list) and used as a source of fruitful ideas, and as input into the formal *Resolve Domain Model* task. They are most easily included as refinements and additions to the FEATURES OF INTEREST. In one sense, therefore, the innovative activities of this task can begin at any point after the domain is selected.

However, this task's placement in the process model emphasizes a more systematic approach to exploring new possibilities in the domain. For this more systematic approach, the following preconditions should be met:

- The domain focus is clear. The *Define Domain* sub-phase helps establish what features belong and do not belong in the domain. Without these criteria, detailed enough to be applied to fine-grained feature descriptions, it is easy, under the guise of extending domain features, to include features that are, properly speaking, outside the domain scope. This is *not* the intent of the task; in fact, such activity could seriously compromise the integrity of the modeling process so carefully maintained in previous activities.

- The INTERPRETED DOMAIN MODEL is complete at least with respect to those features being worked with. Before applying transformations to a feature in the model it should have been cross-checked against the full REPRESENTATIVE SYSTEMS SELECTION. This helps reduce the

number of "dead end" feature combinations that are investigated.

Inputs

- INTERPRETED DOMAIN MODEL. The model of domain features correlated with settings and explanations for patterns of co-occurrence. This is the primary input which is transformed in this task into the DOMAIN MODEL.

- DOMAIN STAKEHOLDER KNOWLEDGE. Additional knowledge may be required to assess the reasonableness of certain proposed feature combinations and potential settings. This relies on broad access to stakeholder knowledge and, to some extent, imagination (what-if scenarios, thinking "out of the box"). This could include access to people who were not informants (in the strict sense of sources for descriptive domain information) and may or may not wind up being specific customers of the asset base.

Controls

- PROJECT OBJECTIVES. Provides focus for selection of particular transformation techniques or areas of emphasis in the task (e.g., optimizing the DOMAIN MODEL for ease of understanding where training of new personnel in the domain is an objective, optimizing for minimal feature-set configurations in a domain where performance constraints are a significant factor.) Perhaps the most important factor is how much innovative results themselves are valued as a possible outcome of the domain engineering process.

- PROJECT CONSTRAINTS. Provides a filter both on level of effort for activities and on the degree to which the model can be extended.

- DOMAIN DEFINITION. Controls the potential expansion of scope of the domain through investigation of feature innovations. Analogy domains and other data from the DOMAIN INTERACTIONS also provides input to certain model transformation techniques.

Activities

➤ Select and apply model transformation techniques

The first set of activities described here are not a linear sequence, but rather form a repertoire of techniques that can be selectively and iteratively applied, either opportunistically, based on perceived characteristics of the descriptive models, or systematically, based on global choices to follow certain techniques to a consistent level of detail for all the models. The specific repertoire of techniques available will vary depending on the domain modeling representations used for the CONCEPT MODELS, FEATURE MODELS, INTEGRATED DOMAIN MODEL and INTERPRETED DOMAIN MODEL. Nevertheless, some general patterns can be identified that should apply quite broadly across representations. The following are suggestions for some of these techniques.

Inversion operations

One possible reason for finding features clustered together is on the basis of perceived semantic relations among the features, such as duality, inversion or opposition. One useful transform involves both identifying opportunities to apply this pattern where there are gaps in current supported features or, alternatively, analyzing certain paired features to see whether some settings might not require both. It can also be useful to distinguish oppositional pairs in general from true inverses in a formal sense, which when executed in sequence return the system to its initial state.

Example. In the Outliner domain, expand and collapse headings, hide and reveal, promote and demote heading are all examples of operations that tend to occur in oppositional pairs. The difference between "hiding" and "collapsing" however does not become apparent until an attempt to "reveal" the hidden sub-structure again. In some cases the decision to support both variants may have significant design impact.

An example of a pairing the terms of which may not be equivalently useful might be "expand all" and "collapse all." If the current outline structure has a single root heading, these features may be identical to "expand all below current heading" provided the user has positioned to the root. But a version of "expand all" could be useful for outlines with multiple root headings, or useful when a user wants to remain positioned at a given location but wants to reveal all surrounding structure. "Collapse all," however, may be used comparatively rarely.

Feature Restriction

In the *Interpret Domain Model* task, regularly co-occurring feature clusters were examined to determine whether the co-occurrence is logically or merely historically based. During *Resolve Domain Model*, clusters identified as historically based can be restricted to elicit new, leaner feature combinations and profiles. Restriction may include suppression of one or more features from a cluster, or replacement of a feature with a specialized variant.

Example. In the Outliner domain, most outlining applications provide both browsing and editing capabilities. This is in part due to the legacy of close connection with text editor applications. In other domains, there is a well-established differentiation between full-featured editing versions and "viewers" which provide light-weight read-only capabilities only. A similar capability may be of value in outlining; for example, if an outlining viewer was used for multi-user collaborative viewing of a common document.

Derive Orthogonal Features

To explore innovative possibilities for any two features, map them as the "axes" of a feature space and check which cells have values filled by exemplar data. Empty cells that represent logically inconsistent combinations of feature values should have been screened out in *Interpret Domain Model*. The remaining cells represent potential feature variants that result from "orthogonalizing" the two features. This basic techniques can be applied to arbitrary numbers of features, but is conceptually the most clear with pairs of features that take discrete values.

In the Outliner domain, there is a distinction between outline structural elements (which can contain sub-structure) and content elements (which can be contained in but cannot contain sub-structure). To derive the orthogonal feature, we would specify "summarize structure item" in an analogous way to "summarize content item." Both operations would then specialize "summarize" in the CONCEPT MODEL for domain operations.

Note that we can more confidently suggest this derived orthogonal feature having done some prior context recovery and interpretation. Since content elements are generally presumed to be text paragraphs in a text-editor environment, an operation to "summarize to first line" is provided for these structures in several outliner exemplars. However, since heading text is generally presumed to be a title or short phrase, the same operation is not usually provided for structure elements. Since we have these correlations to work from, we can also go on to record a hypothesized setting characteristic in which our new feature might be of interest: to wit, any setting where the outlines being worked on are likely to have heading text equally as long as context text.

Orthogonal features will often result from extending an operation to work on a broader class of data (generalizing or shifting the operation).

Feature Shifts Across Binding Sites

In the *Interpret Domain Model* task, feature variants were associated with particular binding sites with respect to various domain settings. Methodically shifting features to other binding sites within the domain-specific system life cycle can suggest previously unanticipated, potentially useful new capabilities.

A typical example of such an innovation would be a multi-algorithm implementation. Rather than compile in a single algorithm to each application, it may be possible to link in various algorithms with differing performance characteristics when a system is configured for a site, or allow for run-time algorithm selection by system users.

The innovation involved need not consist of transforming compile-time to run-time features. For example, the development of spreadsheet compilers came historically after conventional spreadsheet programs. Spreadsheet compilers split the schema definition and data entry tasks into separate, independent components of the system, using a generative step after schema definition to produce a data entry program optimized (and hard-coded) for a particular spreadsheet schema. Data entry personnel were able to work with a smaller, less resource-intensive version of the program that provides necessary functions. Also, central financial offices are able to maintain control over the definition and formatting of spreadsheet schemas, easing the task of merging data from numerous input sources in a consistent way. Thus a restriction of functionality actually resulted in a program configuration more useful than the old one in numerous settings. However, in this case the overall functionality of the product was not reduced. Instead, the functionality was differently distributed within the various operation and usage settings (the schema definers' setting, the data entry setting).

> *Example.* In the Outliner domain, one of the major innovations discovered as a result of modeling concerns the ability to manipulate the open/closed display state of a given outline in more flexible ways.
>
> In *Interpret Domain Model*, we discovered one explanation for the marked variation across outliners in handling display state. In the directory browser environment, where the outliner is a view of separately stored data items, it is possible to associate the current open/closed state locally with the data items (e.g., the files and directories on the file system). This enables the "hide/reveal" feature set: the hide operation changes the display state of only the top heading, and the current display state of the other items is retained with each. However, this same implementation means that when there are multiple views of the same item, these views cannot be in different states.
>
> Word processor-based outliners that use paragraph formats to control the outline display do not have this option. Implementing hide/reveal would be quite difficult given this architectural approach. There are examples of dedicated outline document editors, however, that do provide both "open/close" and "hide/reveal" feature sets. In one case, this is represented to the user not with contrasting operation names, but with contrasting targets for the operations: Expand Topic and Collapse Topic work like Reveal and Hide; Expand Family and Collapse Family work like Open and Close. Since these applications were developed specifically as outliners, this implies that these two distinct features are of high priority from the standpoint of utility.

Use of Analogy Domains

In the *Situation Domain* task, various analogy domains for the domain of focus have been recorded in the DOMAIN CLASSIFICATION. Each of these analogy domains can be revisited to identify analogous features supported in one domain but never carried over to the other. Analogy domains can thus be a source of ideas for both extending and restricting feature configurations.

> *Example.* In the Outliner domain, a "fish-eye" navigation paradigm involves a mode where as one moves to a heading, it is automatically expanded to some level, and the heading one departs is collapsed. From a feature point of view, this represents a composite feature or feature combination (through sequencing). In theory users could manually perform each of these steps (collapse current heading, move, expand new heading) but the effect in terms of the effective metaphor provided by the interface is completely different. In effect, we are creating a tight binding of the display manipulation and navigation operations. We may have identified a tentative analogy domain to outline navigation called "Fish Eye Viewing."

> In *Interpret Domain Model*, we have considered the systems where this feature appears, the presumed utility of the feature for users performing certain kinds of tasks, and its feasibility given certain design decisions. Now we want to consider whether this feature belongs in the final Outliner DOMAIN MODEL. The operative metaphor seems to be equating *structural levels* in an outline with *distance* in viewing real-world scenes; *expanding* as *moving closer* to an object and hence seeing greater detail, *collapsing* as *receding* and losing detail. (An alternative metaphor or analogy would be flying at different heights above a landscape.)

> Given this clarified analogy, how can we define a set of operations, composite operations, default or automated actions and constraints in the outliner interface that serve to create a consistent metaphor for a particular kind of *visual perspective* on the outline structure being viewed? In theory, an outliner application could support different perspective modes of this kind. A conventional outliner, where structural levels can be expanded or collapsed in arbitrary ways, weakens this metaphor. Thus, we discover that to really implement a compelling "fish-eye" capability, we would need to *constrain* the user from making arbitrary display modifications. The modifications should be an automatic consequence of the navigation operations themselves. This results in an innovative feature combination which is noted in the final DOMAIN MODEL.

➤ Close model under selected transformations

Formally validate the models with respect to various closure properties corresponding to model transformation rules described in the section above. Document these as part of the DOMAIN MODEL.

While selectivity in the transform strategies is needed, it is helpful to complete a "pass" through the DOMAIN MODEL with respect to any given strategy at a consistent level of detail, and to document what strategies were applied. This will allow for later incremental application of new techniques.

➤ Resolve feature clusters into feature sets

The purpose of the activities so far has been to extend and reconfigure domain features under a selected set of transformation strategies. This has now provided us enough information to resolve the feature clusters produced in the *Interpret Domain Model* task into the final feature sets that will provide structure within the DOMAIN MODEL. Note that we do not recommend simply applying transformation strategies to individual feature clusters to derive feature sets in a piecemeal

fashion. This would generally result in very fine-grained feature sets but with a lot of redundancy and no overall structural integrity.

As with feature clusters, feature sets can reflect either a configuration of associated features or constraints on combining certain features. In effect, the feature sets should significantly constrain the combinatorics of independently selecting all the individual features specified in the DOMAIN MODEL. The representation strategy for these feature aggregations and constraints depends largely on the Concept Modeling supporting method(s) used for the CONCEPT MODELS, FEATURE MODELS, and INTEGRATED DOMAIN MODEL.

➤ Map feature sets to setting characteristics

Use feature sets as a basis for exploring potential settings where the features would be "of interest" in any of the following respects:

- Potential utility: features would be potentially useful within a setting with the identified characteristics;

- Potential feasibility: features would be potentially feasible to implement within a development setting with the identified characteristics.

This repeats the process of correlating features with settings performed in *Interpret Domain Model*, but extended to innovative features and feature sets. There are several ways of deriving these extended setting characteristics:

- Work directly from elicited STAKEHOLDER KNOWLEDGE.

- By analogy to an established correlation between a feature set and setting characteristics (e.g., derived from interpretation of an exemplar system's feature profile and its setting characteristics): apply the same rationale to hypothesize a correlation between the same setting characteristics and a different feature set, the same feature set and different setting characteristics, or between a different feature set and different setting characteristics.

- Apply similar transformation rules directly to setting characteristics to derive new potential settings. Look for "market extensions" based on the DOMAIN SETTINGS in the DOMAIN DEFINITION. New aspects of existing markets may also be studied. These could include possible undocumented needs in the development or usage settings suggested by particular feature innovations. This can also be effected by generalizing or specializing setting characteristics

    Example. In the Outliner domain, one application setting considered in the DOMAIN SETTINGS might be programmers using outline processors as simple forms of syntax-directed editors for viewing programs. If it takes some work to tailor an outline format useful for a particular language, it is possible that outline templates or format files might become potential reusable assets in and of themselves.

    Note that this raises a domain boundary issue: Is the domain that of outline processing applications, or outline data that can be created using these applications?

➤ Update model and domain boundaries

Reexamine the various workproducts produced to date and validate them with respect to the transformations made to the models. In particular, update the DOMAIN DEFINITION as necessary. For example, features initially modeled within the domain scope that modelers determine weaken the coherence of the model may be *excluded* through boundary adjustments to the various workprod-

280

ucts concerned. Where possible, these features should be shifted over to the appropriate related domain

> *Example.* In the Outliner domain, consider the extension of the domain scope to address out-liner capabilities provided over the World Wide Web. In this environment, there is no real notion of a cursor or cursor position as with a local interface-based outliner application. Yet a number of core outline operations (e.g., promote heading, move sub-heading) may have been implemented using point-drag-and-click interface conventions. In the "resolved" domain def-inition, these conventions are now considered ancillary to the outliner domain proper. The DOMAIN DEFINITION is updated to reflect this. The related domain of "WYSIWYG interface displays" gets assigned a new set of features, and a new layer of outline functionality is defined that is low-level enough to be independent of whether the application runs in a state-less or conventional application environment.

➤ Obtain stakeholder feedback

Validate innovative features through access to DOMAIN STAKEHOLDER KNOWLEDGE via additional interaction with domain practitioners. Reexamine practice in various domain settings from the standpoint of novel feature variants. In particular, consider investigating for the following kinds of data:

- Innovative features may have emerged in engineering change proposals or requests for enhancements.

- Work-arounds or system extensions created by system users that simulate the effect of inno-vative features as obtained through domain modeling.

- For features shifted to an alternative binding site, additional characterization of workflow in the new settings might be needed to determine how useful such a feature would be.

Where the further data acquisition done in the *Interpret Domain Model* task can be viewed as a kind of context recovery or rationale capture to explain descriptive features of exemplars, this can be viewed as exploration or hypothesizing about possibilities with the collaboration of domain stakeholders.

When To Stop

The DOMAIN MODEL can be considered complete enough for the transition to the *Engineer Asset Base* phase when:

- Relevant transformation rules have been applied at a consistent level of detail.

- Results have been propagated back as needed throughout the various workproducts of the *Plan Domain* and *Model Domain* phases.

- The model has been extended to include features that simplify the semantics and coherence of the domain.

- The resulting set of descriptive and innovative features have been configured into feature sets that represent well-designed subsets of domain functionality to be considered by asset base engineers.

- Each feature set has been correlated to setting characteristics where instantiations of domain functionality implementing that feature set would be of definable utility or feasibility.

Guidelines

- <u>Look for transformation patterns in exemplar data</u>. We assume there is a repertoire of transformation patterns that can be applied to the model data. We also assume that the project resources and other PROJECT CONSTRAINTS will not allow for exhaustive application of these transformations. You will not be able to apply all potential patterns and rules; how do you select? Clearly, patterns that appear to promise strategic results in line with PROJECT OBJECTIVES should be emphasized. It is also useful to pick a few complementary techniques to apply.

  A good rule of thumb is to look for the kinds of patterns that already show up in the data and try to apply those patterns consistently across the model as a whole. The current INTERPRETED DOMAIN MODEL may already contain feature combinations that reflect transformations (i.e., more formally, the model is relatively closed under certain transformations because these variants have already been elaborated in domain data). In effect, the more frequently a given transformation pattern occurs in the descriptive model, the more weight can be attached to this rule as a relevant organizing principle within the domain.

  Example. In the Outliner domain, many operations have direct inverses (expand family, collapse family, expand heading, collapse heading, etc.). If a gap in the inverse coverage is discussed, it may therefore be given more weight because the expectation is stronger that the inverse operation should apply orthogonally in the model. If inverse relations are part of the cognitive style of developers in the domain, there should not be a few arbitrary gaps in these relations. On the other hand, we should not gratuitously introduce inversion operations in a domain where this is not very useful. In a domain model with few inverse operations, there would be more opportunities to apply it but less evidence that such application would be perceived as useful by domain practitioners.

- <u>Use techniques that counteract representational assumptions</u>. When features are semantically linked to concepts such as objects, operations and relations, it is possible to derive novel features variants by shifting existing features with respect to a related concept. This technique can yield extremely fine-grained variants. It may separate concepts in ways that conflict with the established software engineering methods used for exemplar systems. For example, where object-oriented approaches advocate binding operations (as methods) to object definitions, maintaining separate object and operation models in the CONCEPT MODELS layer allows for orthogonal combinations leading from either the objects or the operations.

- <u>The domain model is still not prescriptive</u>. Bear in mind that when adding feature combinations to the DOMAIN MODEL modelers are not committing to implement all such feature combinations. The intended scope of feature coverage for the ASSET BASE is determined in the *Scope Asset Base* sub-phase of the *Asset Base Engineering* phase.

  By guaranteeing that adding possibilities to the DOMAIN MODEL does not create commitments for the ASSET BASE the process can add quality and robustness to the model. This can be reinforced by judicious use of techniques like brainstorming and other experimentation. Since some transformation steps will reach dead-ends, selectivity is still required in deciding which innovative features should remain in the final DOMAIN MODEL.

There are extensive opportunities for automated support for this task. If innovation transform rules along the lines of those discussed above could be implemented as patterns to be submitted to a pattern recognition algorithm (along the lines of work in programming and design cliches), then support tools could do semi-automatic weighting of a corpus of innovation rules based on a set of exemplars characterized in terms of a feature model. New innovation rules could also be synthe-

sized by detecting new patterns of common variation in the exemplar set. These are exciting areas
for further research.

7.0 Engineer Asset Base

The *Model Domain* phase of the ODM domain engineering life cycle has produced a DOMAIN MODEL that describes the range of variability for the domain of focus. The model defines a coherent space of possibilities for features in the domain and includes feature profiles that map specific features or feature combinations onto the domain settings (e.g., usage, development) to which they may apply.

As described in the discussion of the Domain Engineering Life Cycle as a whole in Section 4, there are many potential uses for the DOMAIN MODEL produced by the first two phases of ODM. For example, the DOMAIN MODEL can be used:

1) as a source of codified domain knowledge;

2) as a more formal basis for specifying or building individual systems in the domain; or

3) as a basis for developing an asset base that addresses the needs of multiple systems.

The *Engineer Asset Base* phase of ODM is designed to focus specifically on (3) above. Its purpose is to scope, architect, and implement an ASSET BASE that supports a subset of the total range of variability encompassed by the DOMAIN MODEL, a subset that addresses the domain-specific requirements of a specific set of customers.
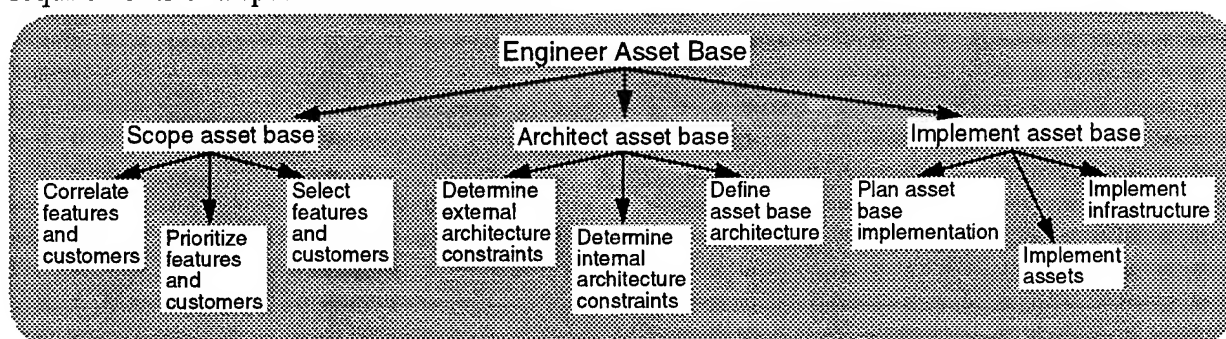
Exhibit 76. Engineer Asset Base Process Tree

The key benefit of the ASSET BASE produced in this phase of domain engineering is an overall economy in the cost of developing systems in the domain, when viewed from the perspective of the customers for the ASSET BASE taken as a whole. The key challenge of this phase is to identify an appropriate market of customers and the features required to support this market, and to make the ASSET BASE a viable and ongoing structure for achieving increasing levels of reuse. An additional challenge in the *Engineer Asset Base* phase is to manage the potentially explosive variability in the domain in a process that eventually produces tractable specifications for implementors to create the asset base infrastructure and individual assets.

Approach

The concept of an *asset base* can be difficult to convey, but not because asset bases themselves are unusual or arcane. In fact, software asset bases are rather commonplace, found under a host of different names and developed using a variety of implementation techniques. Some examples of asset bases are:

• A library of vendor- or operating system-specific system services;

• An object-oriented framework with associated classes, methods, etc.;

285

- A domain-specific kit, including application generators, templates, etc. as in [Gris93];

- An ORB-based component base or library, together with documentation, access mechanisms, and configuration management policies;

- Application-specific languages or 4GLs (e.g., PowerBuilder, GUI-builders, etc.) particularly those focused in specialized niches or business areas;

- A domain-specific software architecture (DSSA).

One might ask why a new term — "asset base" — is needed to refer to these familiar engineering examples, and why a special process — *Engineer Asset Base* — is needed to produce them. First, despite their ubiquitousness, there is no generally accepted process for building these kinds of reusable resources. The process is different from (a) developing a single application with a single set of requirements, (b) building a product targeted for end-use (e.g., a text-editor or tax accounting program), in that it may involve developing capabilities that support a variety of different end-uses, and (c) developing "conventional" reusable components available for reuse more-or-less independently and opportunistically. Second, there is no standard term by which we can refer to this as a discrete class of engineering problems. New concepts, terminology, and processes are needed that encompass this class of problems and offer approaches for solving them.

The *Engineer Asset Base* process described in this section is predicated on several assumptions. If not all of these conditions are satisfied, somewhat simpler variations of the process can be tailored to suit the circumstances, or alternative uses of the DOMAIN MODEL, as listed above, can be pursued. The assumptions are:

1) Potential customer "value chains" exist involving at least the following three distinct participants, or settings:

   a) Asset base developers;

   b) Application developers who will use assets from the asset base in building their applications; and

   c) End-users of the applications.

   There may be additional links in the chain, such as an asset base manager who serves as an intermediary or broker between asset base developers and application developers. The key point here is that the "customer" relationships in an asset base context can be substantially more complex than for an individual system.

2) Multiple asset instances or configurations will be required to satisfy the varied needs of different applications.

   The ODM *Engineer Asset Base* process is designed specifically to support variability in domain functionality within targeted applications. It assumes that the needs of these applications are likely to be filled by tailoring or configuring a set of assets in specialized ways, based on the anticipated variability in the applications. The asset base forms a coherent, adaptable whole rather than a set of static, uncoordinated components that are reused as opportunities arise. For example, in the case of a GUI-builder, different applications may require very different GUIs, but the variation across the different systems can be captured in an asset base and supported by a generative tool that makes the proper configuration decisions to generate each desired implementation. Asset base engineering involves designing the supply mechanism where multiple asset configurations will be required. The mechanism could be a library of interdependent components, application generators or composers, or

other alternatives or hybrids.

3)   A domain model exists that will form the technical basis for building the asset base.

In order for a set of assets to be reusable in predictable, repeatable ways, there must be some frame of reference for understanding the assets' intended scope of applicability. This frame of reference must define the range of variability the assets must address and the interdependencies and constraints they must satisfy. The DOMAIN MODEL developed in *Model Domain* is intended to provide this frame of reference, and the presence of a model such as that is a necessary condition for engineering an ODM asset base. The scope of applicability of each asset can be defined as a subset of the overall feature space encompassed by the domain model, and the scope of different assets may overlap (i.e., as variant components rather than structural decomposed functions as in system engineering). The collection of assets may or may not cover the entire domain space, but must be consistent with the definition of that space in the domain model.

4)   ***Economies of scope*** exist that justify implementing a set of assets as a coherent asset base, rather than separately.

To warrant an asset base engineering process, there must be identifiable economies of scope in implementing the set of assets as a coherent collection. That is, developing the asset base as a coherent whole must be cheaper, more effective, more scalable, or yield more sharable results than developing a group of assets separately, in a less coordinated manner. For example, economies of scope can involve the following factors:

—   An ***asset base architecture*** that clearly defines the rules and constraints for composing or generating assets to support domain functionality within multiple applications.

—   Sharing and internal reuse among the assets within the asset base.

—   Shared economies in the development process for the various assets. This includes incremental development strategies (e.g., successive versions with enriched capabilities, or a superset implementation with masked functionality).

—   Shared asset and application infrastructure (for example: library mechanisms; ORB interfaces; generative technology; organizational structures for asset base management, brokerage, and promotion).

A key aspect of the *Engineer Asset Base* process is to identify economies of scope within the domain and the organization, analyze trade-offs among them, and exploit those yielding the greatest benefit relative to cost when architecting and implementing the asset base and infrastructure.

Whereas the *Model Domain* phase is primarily *descriptive*, the *Engineer Asset Base* phase is *prescriptive* in nature. In this phase domain engineers commit to a specified range of functionality to be provided by assets developed/adapted for the asset base. Not all feature combinations noted during domain modeling need to be supported; some may not be of sufficient general use to warrant inclusion in reusable assets. Conversely, as a result of the *Resolve Domain Model* task that concludes the *Model Domain* phase, features and feature combinations may be selected for the asset base that did not occur in any representative studied (i.e., in no existing system or known set of requirements for new systems).

If the purely DESCRIPTIVE MODELS produced by *Describe Domain* describe the "as is", and the DOMAIN MODEL emerging from *Resolve Domain Model* describes what "could be", the ASSET BASE MODEL developed in *Scope Asset Base* describes what "will be." That is, it describes the

287

Exhibit 77. Engineer Asset Base IDEF$_0$ Diagram
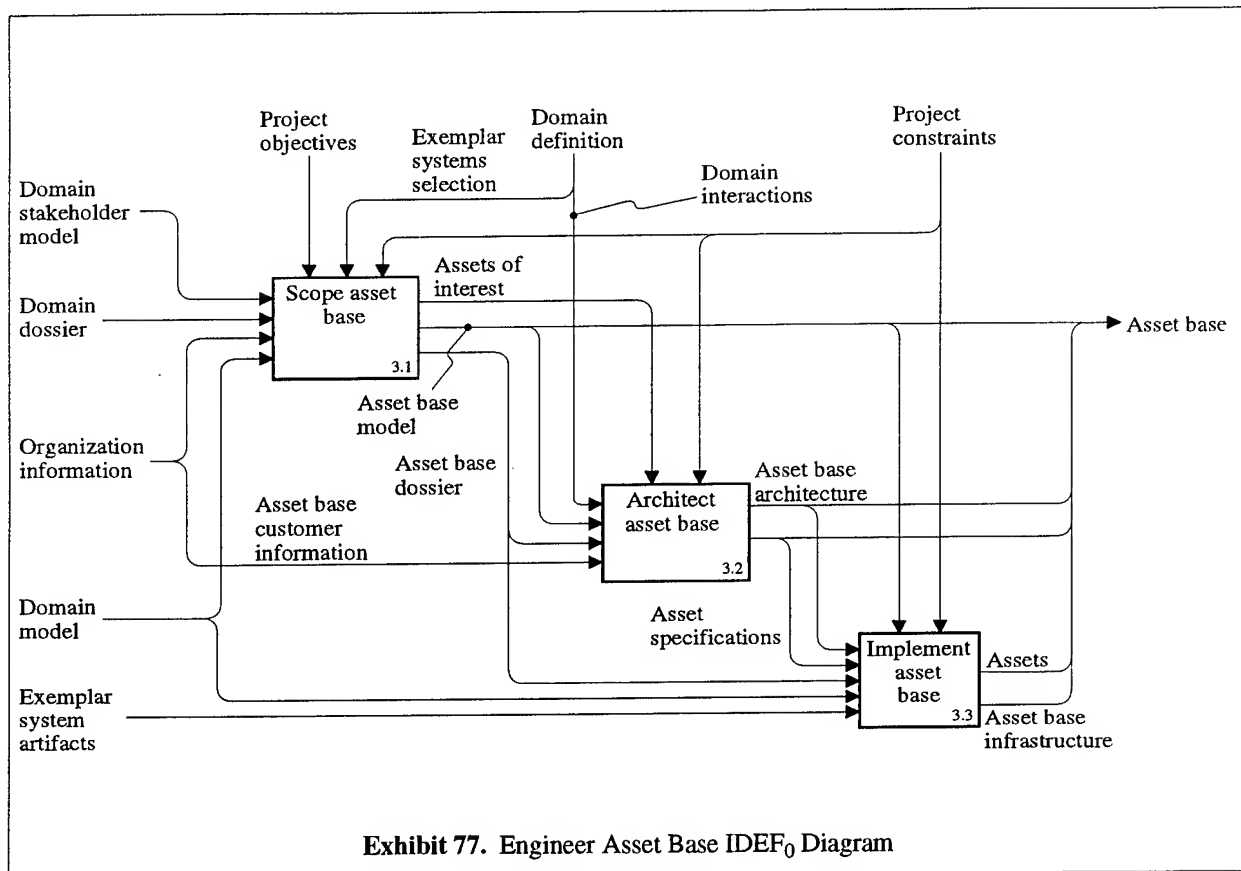
asset base's intended scope to meet specific customer or market expectations. The *Model Domain* phase has produced a mapping from exemplar artifacts, to conceptual entities, to semantically characterized features. These features and feature combinations form the baseline for determining the range of features to support in the ASSET BASE.

These features to support are captured in the ASSET BASE MODEL by *Scope Asset Base* and are eventually clustered and allocated to individual ASSETS within the ASSET BASE ARCHITECTURE by *Architect Asset Base*. The ASSET BASE ARCHITECTURE also captures the interrelationships and constraints among the ASSETS to support asset composition and reuse. The ODM concept of an *asset base architecture* is different from, though clearly related to, a conventional *system architecture*.

An asset base architecture must capture the functional and structural variability within an asset base and must support resolution of that variability to produce specific instances of domain functionality that can operate within target applications. The asset base architecture needs to encompass the function and structure of all valid system artifacts (including system architectures) to be produced using the asset base.

Results

The fundamental result of the *Engineer Asset Base* phase is an ASSET BASE that is applicable to the development of a set of application systems that are clearly scoped in terms of the domain functionality they support. The ASSET BASE is a composite workproduct consisting of the following workproducts:

- An ASSET BASE MODEL that defines the features and feature combinations to be supported by the asset base for selected *customer settings*.

- An ASSET BASE ARCHITECTURE that models the range of variability among system architectures or configurations that can be derived from feature combinations in the asset base. It includes the rules, relationships, and constraints among assets that govern how they can be combined and instantiated to produce application artifacts supporting specific feature ensembles.

- A set of ASSET SPECIFICATIONS defining the required features, interfaces, and behaviors of each ASSET for implementation purposes.

- A set of ASSETS that are the directly reusable entities within the ASSET BASE. In general, an *asset* can be considered any component, tool, or other resource that helps domain practitioners do their work more effectively. In the system and software engineering context, an asset is a discrete system/software entity that encompasses some set of features (typically, feature variants) and can be instantiated to supply those features (or a subset) to one or more application artifacts.

- An ASSET BASE INFRASTRUCTURE that supports application engineers in using the ASSET BASE to develop and maintain their applications.

The ASSET BASE represents the ultimate output of the domain engineering process (assuming that the domain engineering objectives included development of an asset base, rather than just, say, a domain model). All the work done throughout the process culminates in this set of reusable resources.

Process

As shown in Exhibit 77, the *Engineer Asset Base* phase consists of three main sub-phases:

- In *Scope Asset Base* domain engineers begin the task of directly specifying functionality to be supported by assets in the asset base. The focus accordingly shifts from *domain informants* to potential *customers* for the assets. Both the intended customers and the feature variants to be supported by the asset base are selected in parallel. Feature variants are prioritized for utility relative to a specific set of potential *asset base customer* settings, and for feasibility relative to available asset implementation and management infrastructure.

- The primary goals of the second sub-phase, *Architect Asset Base*, are: (1) to produce an ASSET BASE ARCHITECTURE flexible and adaptable enough to cost-effectively cover the desired range of variability for the asset base, and (2) to define this architecture so that it is evolvable, maintainable, and minimally impacted by different implementation choices for different assets and by evolution of assets over time.

  Selected features and feature configurations are mapped to high-level ASSET SPECIFICATIONS. These specifications may reflect significant restructuring from the system artifacts first studied descriptively in *Model Domain*.

- The third sub-phase, *Implement Asset Base*, implements both the assets and the required infrastructure for the asset base. A key challenge in this sub-phase is performing the trade-off analysis to determine the best implementation strategy (e.g., component- or generator-based techniques) for each asset and each configuration of assets that is likely to be accessed as a whole.

More than the preceding phases, the *Engineer Asset Base* phase relies heavily on supporting methods to achieve its objectives. This is because the *Engineer Asset Base* activities are more closely related to "conventional" software planning and engineering than the previous phases. *Scope Asset Base* relies on market analysis supporting methods to determine the marketplace and technical scope for the asset base, *Architect Asset Base* relies significantly on architectural analysis and design methods for structuring the asset base, and *Implement Asset Base* relies strongly on component, generator, and tool implementation and validation techniques, as well as conventional software development planning methods. As with all supporting methods, but perhaps more so in this phase The intent here is to allow ODM practitioners to employ detailed methods with which they are most familiar or comfortable. This is true in general for all ODM supporting method choices, but perhaps more so in this phase because of the greater overlap with processes an organization is already likely to be performing.

Guidelines

- Use a phased engineering strategy. Ignoring opportunities for variability will simply back developers into corners and limit possibilities for later evolution. Conversely, attempting to engineer the asset base to satisfy too many dimensions of variability simultaneously will exceed the capacity of current techniques. An incremental strategy provides a middle ground—building assets of limited scope in the context of a long-term plan for evolving towards greater variability over time.

- Apply generative techniques where possible. Generative techniques are a classic strategy for managing variability. When a subroutine has gained too many parameters with cross-constraints and dependencies, it can often be replaced by a generative solution that is much easier to use and can generate optimized solutions as well. In principle these same techniques can be applied to the configuration or "gluing" of large-scale components into aggregates. This approach will break down, however, if variability needs to be supported at multiple, interacting structural levels simultaneously. In this case, the problems of tractability re-assert themselves.

- Relationship between *Scope Asset Base* and previous phases.

  — The ASSET BASE MODEL should remain a *subset* of the final DOMAIN MODEL. New feature variants or types of customer settings should be migrated back to the INTERPRETED DOMAIN MODEL; and filtered through the *Resolve Domain Model* task. The iteration might require backtracking as far as the DOMAIN DEFINITION.

  — The asset base customers should remain a *subset* of the domain stakeholders. Asset base customers may be identified who were not originally considered as domain or project stakeholders. The analysis of the domain marketplace in *Scope Asset Base*, coupled with the broadened understanding of the domain in *Refine Domain Model*, may yield a set of customers that was not envisioned during the original stakeholder analysis. These customers and associated settings should be migrated back into the DOMAIN STAKEHOLDER MODEL.

Note that these guidelines focus on consistency maintenance and traceability across workproducts and phases. There may be circumstances (e.g., ODM is being applied rapidly for evaluation or prototyping; no evolution of the asset base is envisioned) in which such consistency and traceability are not important. In such situations, the above guidelines should be treated as exceptional conditions. Even when traceability is considered important, there is a risk that iteration between the *Refine Domain Model* and *Scope Asset Base* sub-phases will result in an out-of-control process: i.e., an innovative feature is conceived, a potential customer is found, the customer settings are studied, opportunities for other features are discov-

ered, the innovation process continues, and so forth. In this way the set of features and proposed span of customer settings could spiral out of scope. One risk mitigation strategy is to strictly preserve the sequence of descriptive, interpretive, innovative, and prescriptive modeling stages, and to limit the feedback cycles allowed.

- Relationship between *Scope Asset Base* and *Architect Asset Base*.

  — The ASSET BASE ARCHITECTURE can reflect a *subset* of the ASSET BASE MODEL. Architectural constraints may rule out support for the full range of variability suggested in the feature and customer profiles in the ASSET BASE MODEL. Expect some iteration between the *Scope Asset Base* and *Architect Asset Base* sub-phases.

  — Consider prototyping the ASSET BASE ARCHITECTURE before completion of the ASSET BASE MODEL. While the final ASSET BASE ARCHITECTURE should be validated against selected features in the ASSET BASE MODEL, trial architecture efforts can be initiated earlier to test the suitability of a particular architecture formalism or architecture description language.

  — Consider re-scoping the asset base if architectural constraints are prohibitive. In general, the intent in *Architect Asset Base* is to resolve architectural constraints to establish requirements for variability that can be met with a net economy of scope or effort. In the worst case, where the constraints imply a need to build completely separate variants to address each distinct feature set or customer setting, it may be advisable to return to *Scope Asset Base* and re-scope accordingly.

- Relationship between *Architect Asset Base* and *Implement Asset Base*.

  — Defer consideration of opportunities to reengineer legacy artifacts until *after* a coherent architectural approach is determined. Facilitating this reengineering is a goal of the ODM process, and significant effort goes into preserving information that will support it. But, in principle, such reengineering actually re-introduces an element of *ad hoc* or opportunistic reuse into the heart of the *Engineer Asset Base* phase of domain engineering. Allowing design decisions to be guided primarily by these concerns would risk undoing much of the value derived from earlier steps of the process. The process recommended here embodies a sequencing strategy designed to mitigate this risk.

  — Consider prototyping some individual ASSETS before completion of the ASSET BASE ARCHITECTURE. These should be considered prototypes, implemented as a risk reduction strategy in order to validate a candidate technology. While technology selection decisions are deferred where possible until the *Plan Asset Base Implementation* sub-phase, in practice the feasibility of applying some technologies will require some lead time to set up the infrastructure, train people and obtain required organizational commitment. Doing trial implementation as early as possible therefore makes good sense.

    This may appear in conflict with the previous guideline. The intent is that although some reengineering may be done during prototyping, such reengineering is intended to feed the ASSET BASE ARCHITECTURE design process. Final decisions about how to reengineer legacy artifacts into assets should be made only after relevant architectural decisions.

  — Architecture and technology selection may be parallel. The objective in the ODM process model is to encourage definition of a relatively technology-independent asset base architecture. In practice, architecture development and technology selection may be closely interleaved. Technology choices will require tuning and adjustment of the architecture; architectural choices may impose significant constraints on technologies selected.

— <u>Multiple *Implement Asset Base* sub-phases may be appropriate</u>. Once ASSET SPECIFICA-
TIONS are determined, subsequent *Implement Asset Base* sub-phases can implement all
or some of the specified ASSETS. In addition, asset utilizers can implement some assets
for which there are only specifications.

7.1 Scope Asset Base

The primary purpose of this sub-phase is to derive an overall feature profile for the ASSET BASE and to identify and characterize the *market* for the asset base in terms of the settings in which ASSET BASE customers (i.e., application developers) will potentially utilize domain assets. The key function of this sub-phase is to derive a subset of the features and setting profiles described in the DOMAIN MODEL, mapped to a specific set of customers for the asset base.
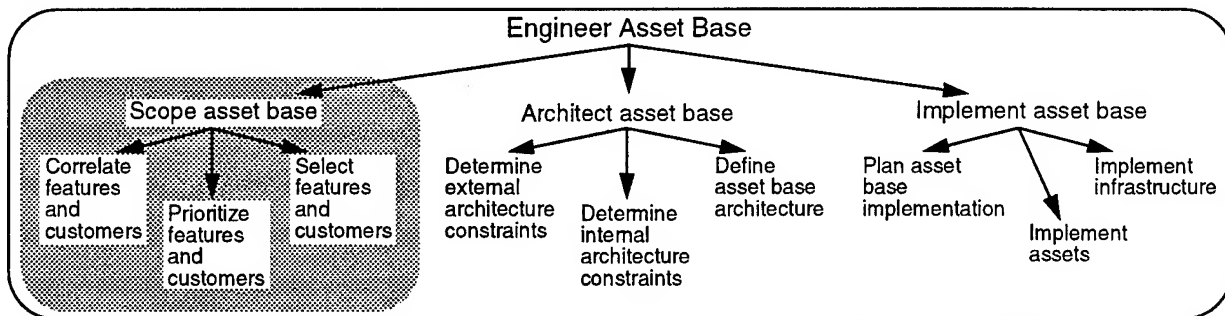
Exhibit 78. Scope Asset Base Process Tree

A key challenge of this sub-phase is that both the features and the customers to be supported are dynamic, and feature and customer decisions are interdependent. Another challenge is the fact that the correlation of features to customer settings is initially based on the descriptive correlations in the DOMAIN MODEL; hence, there is considerable uncertainty in how well-matched the correlations are to the real settings in which reusable assets must be adopted. To address these challenges and mitigate the associated risks, *Scope Asset Base* involves (a) market analysis based on the notion of *value chains* that characterize producer-consumer relationships between a series of stakeholder settings, beginning with the asset developer and ending with the application end-user, and (b) analysis of feature utility and feasibility relative to specific settings.

Approach

The *Scope Asset Base* sub-phase can be compared to the requirements analysis phase of conventional software engineering. The resulting ASSET BASE MODEL specifies what features must be supported by the asset base; in subsequent sub-phases, the ASSET BASE ARCHITECTURE and ASSET IMPLEMENTATION PLAN will embed specific technology choices that determine how these feature requirements will be satisfied.

However, this sub-phase also differs significantly from the requirements analysis phase in conventional software engineering in that:

- features, unlike requirements, can impose constraints on design and even implementation details; they are not just high-level requirements;

- features within a feature set can conflict, and features across feature sets can be redundant (i.e., feature sets are not *partitionings* of the total range of features in the same way that a requirements allocation partitions a set of system requirements); and

- there is no pre-ordained single customer to whom all features apply.

Domain modeling has produced a set of features, profiles of the kinds of settings to which the features might apply, feature interdependencies and, possibly, feature profiles of domain artifacts that may become candidates for reengineering into assets. The task now is essentially one of simulta-

neous constraint satisfaction. In the end, a set of features to be supported by the asset base must be specified. The individuals and organizations who are anticipated to use the asset base to build application systems must also be selected. They will be the potential *asset base customers*.

To aid in selecting asset base customers, the "business opportunities" involving the asset base are analyzed. The ODM notion of business opportunity analysis involves applying a set of domain engineering-oriented principles and guidelines to supplement conventional market analysis techniques (i.e., to supplement the Market Analysis supporting method area on which Scope Asset Base relies).

A key aspect of ODM business opportunity analysis is the discovery and analysis of multi-tiered *value chains*. In selecting asset base customers, the focus is on matching potential *customer settings* with feature combinations that are candidates for inclusion in the ASSET BASE MODEL. Although the asset base customers (i.e., those who are direct consumers of the asset base, typically to build end-user applications) are ultimately the most important to consider in determining features to support, the needs of those customers can only be determined within a broader business context characterized in terms of multi-tiered value chains. These value chains involve not only the direct asset base utilizers, but also the asset base developers themselves, end users of applications built from the asset base, asset base brokers who support and encourage use of the asset base, and perhaps other kinds of customers who are once or twice removed from the asset base but impact what it contains, how it is developed, or how it is used.

Studying value chains is critical in ODM because (a) the feature sets in the DOMAIN MODEL are defined in terms of kinds of settings to which they are intended to apply, and similar, more concrete correlations should be made in scoping the asset base, and (b) the value chains reveal distinct patterns of supply and demand and stakeholder interdependencies that will help crystallize requirements for the asset base. Value chains support prioritization decisions (e.g., between the

Exhibit 79. Scope Asset Base IDEF$_0$ Diagram
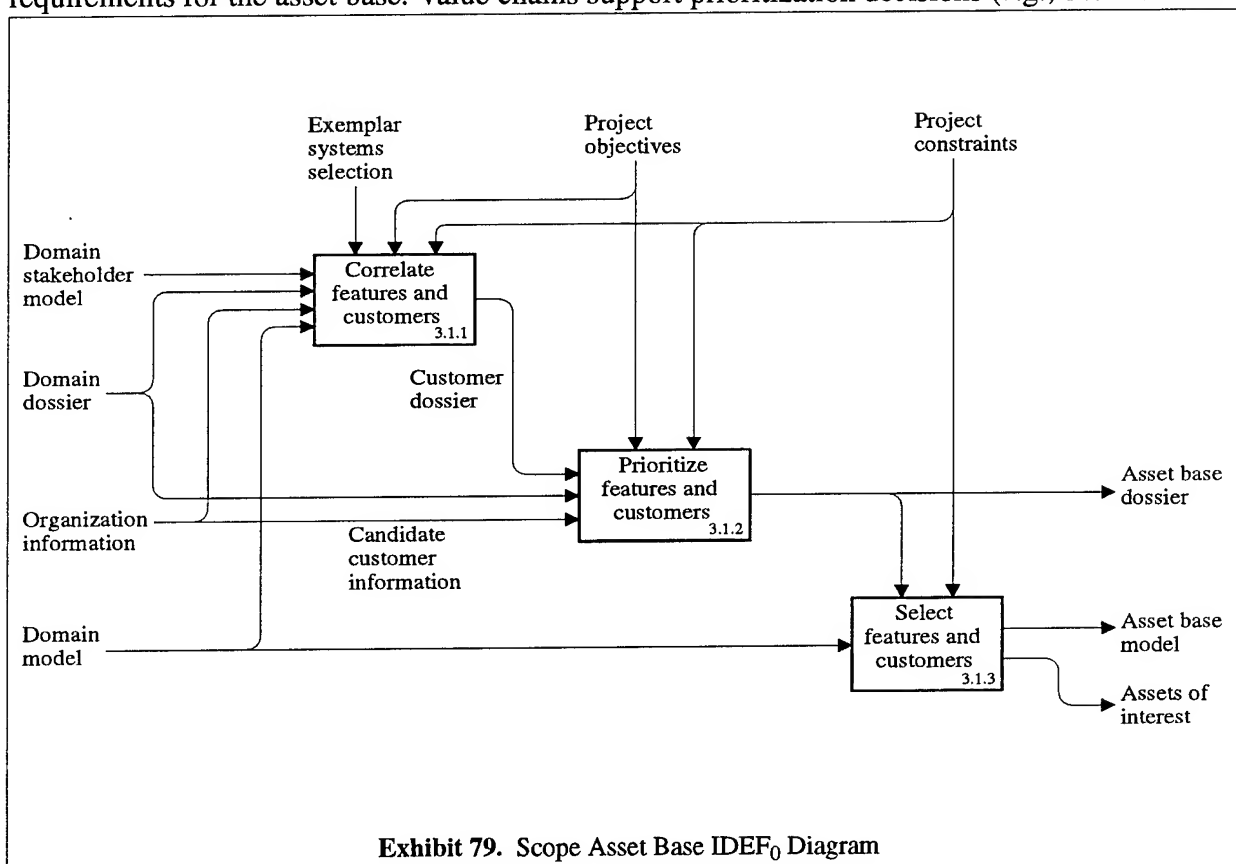
number of end-users satisfied by a given feature versus the cost of implementing it) by enabling tradeoffs between the end-users' interest in using certain features and the developers' interest in implementing them.

Results

The primary results of the *Scope Asset Base* sub-phase are:

- The ASSET BASE MODEL that defines the features and feature combinations to be supported by the asset base for selected customer settings.

- The ASSET BASE DOSSIER that characterizes customers and settings relevant to the asset base, relates these customers and settings to their features of interest, and prioritizes those features of interest relative to criteria such as customer importance, utility, and feasibility.

The ASSET BASE MODEL can be viewed as defining the requirements for the next sub-phase, *Architect Asset Base*, while the ASSET BASE DOSSIER provides supplementary information to aid in building the ASSET BASE in accordance with those requirements.

Process

As shown in Exhibit 79, *Scope Asset Base* consists of three main tasks:

- The *Correlate Features and Customers* task maps the feature sets and associated setting profiles in the DOMAIN MODEL to specific candidate customers or classes of customers, derived from the DOMAIN STAKEHOLDER MODEL or other sources. This involves identifying asset base business opportunities in terms of value chains linking various candidate customer settings.

- The *Prioritize Features and Customers* task evaluates the anticipated benefit of each feature set among the candidate customer settings and the anticipated level of effort to implement and maintain the feature sets to support those settings.

- The *Select Features and Customers* task completes the transition from descriptive to prescriptive modeling, from the realm of *possibilities* to that of *commitments*. In this task, the priorities are further evaluated relative to business and technical objectives and realities, customer and feature conflicts are resolved, and a commitment is made to support specific features and customers in the asset base.

At the end of this sub-phase, there should be traceability links demonstrating consistency between anticipated asset customers and supported features, and between architectural and more fine-grained features to be supported.

Guidelines

- <u>Business opportunity analysis may be of value even when asset base customers seem pre-determined</u>. In some cases, going through the full-blown *Scope Asset Base* process may not seem justified. An organization may begin the domain engineering process with a very clear idea of who the asset base customers will be: a set of current and future product engineering projects supporting a product line within the organization. In fact, those customers may have had to be established in order to sell the idea of domain engineering to management. It still will be necessary to determine which features to support for which customers, so some aspects of *Scope Asset Base* must still be performed, but there may appear to be little need to

do a business opportunity analysis. However, even under such circumstances, it may be useful to develope additional business scenarios to consider broadened or alternative markets.

- Prioritize specific customers carefully. Being clear about the organizational settings that the asset base needs to support is essential to any systematic approach to the problem. Designate specific customers; but *more than one*. The added awareness of contextual dependencies and potential variability from building to support two or three customers is dramatically greater than for a single customer; and far more feasible than building for "all customers" in general (but none in particular).

- Consider shifts in current organization structure and business relationships. Often asset base scoping will involve decisions that involve possible shifts of business relationships between companies; i.e., a company being in a service relationship (building applications for an end-user) versus providing a product that *helps* their customer build applications. Similarly, asset base scoping may often require the creation of new business entities, either internal or external to a given organization.

- Feature-driven vs. customer-driven. This sub-phase can be performed in either a *feature-driven* or *customer-driven* sequence, or in a combination of the two. The feature-driven approach works outward from identified sets of features to associated potential settings to specific customers that match the profile of those settings. Alternatively, if there are definite *core customers* mandated or strategically determined for the project, these customers can be used to establish a corresponding set of *core features* to support.

- Bottom-up vs. top-down. Selection of prescriptive features can also be performed in two general ways, using a *bottom-up feature-driven* versus a *top-down, architecture-driven* approach. The bottom-up feature-driven approach describes feature variants to be supported for individual asset-level functionality. The architecture-driven approach explores feature sets describing the domain system as a whole.

Trade-off analysis may iterate between bottom-up analysis of individual features and feature variants, and top-down analysis of feature profiles for entire domain systems (i.e., systems or subsystems representing the full scope of the domain of focus, rather than individual features within the domain).

7.1.1 Correlate Features and Customers

A broad goal for this first task within the *Scope Asset Base* sub-phase of *Engineer Asset Base* is to establish an overall organization context for the *Engineer Asset Base* phase. Assumptions about the context for the ASSET BASE that were made at the start of the project, in the *Plan Domain* phase, are almost certain to have changed over the course of the *Model Domain* phase. In fact, one criterion of success for this phase is that these assumptions will have changed. Domain practitioners have participated in the modeling effort and should perceive possibilities for reusable assets in different terms; modelers should more thoroughly understand the limitations and possibilities of the domain.

As a result, the relevant context for the DOMAIN MODEL may be quite different than for the ASSET BASE. In fact, multiple asset bases could be engineered using a common DOMAIN MODEL as a basis. In order to reap the benefits of this new situation, a critical element of the ODM process is to explicitly re-contextualize as part of this task.

The DOMAIN MODEL includes a number of overall *feature sets*, each representing a coherent segment of domain functionality, and characterized in relation to other feature sets and to profiles of potential settings to which the feature sets might apply. This task focuses on correlating these fea-
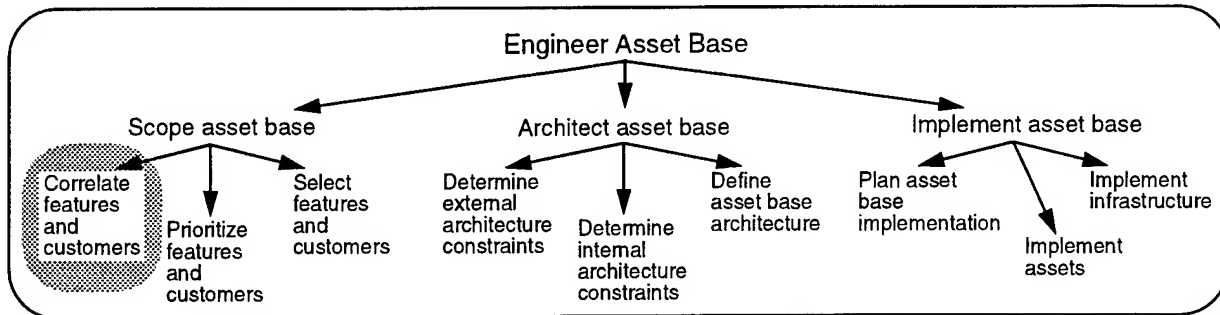
Exhibit 80. Correlate Features and Customers Process Tree

ture sets and setting profiles with specific customers. The primary result will be a CUSTOMER DOSSIER in which feature sets that are viable candidates are identified and correlated to candidate *customer settings* within the overall stakeholder context for the domain and the project.

Approach

The *Correlate Features and Customers* task is analogous to the early context-setting activities in the *Plan Domain* phase, i.e., the *Determine Candidate Project Stakeholders* task within the *Set Project Objectives* sub-phase. In both these tasks, project planners and asset base modelers respectively perform a descriptive scan of the possible stakeholders, deferring to later tasks the final selection and commitment. However, the actual process performed is quite different in this task, because it builds on an already completed DOMAIN MODEL.

The focus in this task is on constructing initial candidate business scenarios expressed in terms of value chains. Many of the candidate customers involved in these scenarios will typically be drawn from the existing Domain Stakeholder Model, but to the extent that the intervening Define Domain and Model Domain processes have yielded new insights about the nature of the domain and its applicability, it may be useful to consider customers who were not originally viewed as stakeholders.

Trade-off analysis and selection of which feature sets to support and which markets to target as explicit customers are performed in the subsequent tasks of this sub-phase, *Prioritize* and *Select Features and Customers.*

Workproducts

■ CUSTOMER DOSSIER

This workproduct captures the following information:

* Initial characterizations of the candidate customers and the domain-relevant settings in which the customers are involved.

* Interrelationships between customer settings, expressed in terms of value chains associated with candidate feature sets derived from (i.e., a subset of the features in) the DOMAIN MODEL. These value chains can be viewed as a set of candidate business scenarios or business opportunities for the asset base. The BUSINESS OPPORTUNITIES TABLE worksheet template in Exhibit C-12 suggests a format for expressing this information.

* A correlation (or mapping) between customers/settings and candidate feature sets, reflecting

the business scenarios. This mapping shows the anticipated degree of interest that the candidate customers (in their relevant settings) have for the candidate feature sets. The mapping is *many-to-many*: Each feature set may be deemed of interest to multiple customers. Conversely, each customer may be interested in multiple feature sets.

— For feature sets that are **components** of the domain, the more components that interest a given customer the more likely it is that the customer will utilize multiple assets in subsystem level aggregations.

— For feature sets that are **specializations/generalizations** of the domain, a customer's interest in more than one feature set indicates the degree of variability that is required in the ASSET BASE.

The FEATURE/CUSTOMER MATRIX worksheet template in Exhibit C-13 suggests a format for expressing this correlation information.

When to Start

- There is a complete and validated DOMAIN MODEL for the domain of focus. In particular, the number of possible feature variants must have been reduced through constraints documented in the INTERPRETED DOMAIN MODEL and transformations of features incorporated into the final DOMAIN MODEL.

- There is a commitment to proceed with the *Engineer Asset Base* phase of domain engineering. This criterion allows for the possibility that some delay may ensue between completion of the DOMAIN MODEL and the *Engineer Asset Base* phase. Much of the care that goes into the completion and validation of the DOMAIN MODEL is to assure its viability over time. Also, as mentioned above, multiple *Engineer Asset Base* efforts could be initiated starting from a common DOMAIN MODEL.

Inputs

- DOMAIN STAKEHOLDER MODEL. The primary source for candidate customers for the asset base. The *Correlate Features and Customers* task works with the *domain* stakeholders, not the original *project* stakeholders. Project stakeholders without explicit interest in the domain results are still involved and reflected in the controls on this task, i.e., PROJECT OBJECTIVES/CONSTRAINTS.

  The *Correlate Features and Customers* task works with all domain *stakeholders* (and perhaps other newly-discovered potential customers) and not only domain *informants*, the subset of stakeholders who were the focus of interactions in the *Model Domain* phase. The aim is now to cast the net as wide as possible; some informants will become customers, but many potential customers will not have been consulted as informants. How well the feature sets within the DOMAIN MODEL meet the needs of these other customers will be one significant validation of the quality of the sample selected in the *Plan Data Acquisition* task of the *Acquire Domain Information* sub-phase.

- DOMAIN MODEL. The source for candidate feature sets for the asset base, as well as for the associated setting profiles that guide identification of specific candidate customers.

- DOMAIN DOSSIER. Includes information about exemplar systems and associated settings that is useful for eliciting candidate customers and settings and correlating those settings with particular features.

298

- ORGANIZATION INFORMATION. Information about the organization's business environment which helps to clarify the organization's customers, product lines, and business prospects.

Controls

- PROJECT OBJECTIVES. Overall project objectives, specific objectives for the *Engineer Asset Base* phase that were identified earlier in the project, and any modified or new objectives, possibly a result of or reaction to earlier project phases.

- PROJECT CONSTRAINTS. There may be new constraints, including constraints created by previous phases of the domain engineering life cycle; e.g., success in the *Model Domain* phase may engender expectations for the *Engineer Asset Base* phase of the project.

- EXEMPLAR SYSTEMS SELECTION. The list of systems that intersect the domain, from which candidate customer systems will be drawn in this task.

Activities

➤ Develop business scenarios

Develop a set of business scenarios for the asset base involving multi-tiered value chains. Record the results in the CUSTOMER DOSSIER. The BUSINESS OPPORTUNITIES TABLE worksheet template in Exhibit C-12 suggests a format for expressing this information.

There are various strategies for discovering value chains. The options can include:

- Working from the beginning of the chain to the end (i.e., from the features/sets, work forward to identify application developers who might want to provide those features in applications and identify customers they might want to target),

- Working from the end back towards the beginning (i.e., from market profiles or other characterizations of end user needs, identify application developers who might want to satisfy those needs),

- Working from the middle out (i.e., from a particular developer perspective, identify end user customers within their perceived/desired marketplace and asset base features/sets that could contribute to meeting the customer's needs effectively).

In general, in identifying candidate customers and settings when applying these strategies, use the DOMAIN STAKEHOLDER MODEL to identify stakeholders with potential interest in the asset base. Using the PROJECT OBJECTIVES, identify those who are *core customers*: i.e., project success depends on satisfying their requirements as asset base customers. Also consult ORGANIZATION INFORMATION to determine if there may be other potential customers consistent with the organization's business strategies who were not considered during domain planning. For all candidate customer settings, indicate the *stakeholder interests* that seem relevant to requirements for the asset base.

Use the DOMAIN MODEL to identify sets and help in evaluating their viability for consideration. This could include the entire model if there are no semantically significant ways of excluding feature sets. Useful filtering criteria will likely emerge as specific strategies such as those listed above are employed. For large and complex models, the PROJECT CONSTRAINTS should provide some filtering criteria; e.g., global contextual constraints that exclude certain feature variants and combinations. Also, the EXEMPLAR SYSTEMS SELECTION and DOMAIN DOSSIER can be used to

identify features present in exemplar systems that are known to be relevant to one or more candidate customer settings. Such features should be strongly considered as candidate asset base features.

The data collected here may be dynamic and time-sensitive; this highlights the importance of this task being performed close to the time when asset base development will begin.

➤ Correlate feature sets with candidate customers

Depending on the strategies used above, the candidate value chains may already correlate feature sets and candidate customer to a significant degree. Even if this is true (and especially if it is not), further analysis should be performed, both to explore further relationships among stakeholder settings and features than was initially apparent and to ensure that information in the DOMAIN MODEL and DOMAIN STAKEHOLDER MODEL has been fully exploited in defining the value chains.

A critical technique here is to enrich the value chains (and perhaps generate new ones) by varying the *feature binding sites* for features in the feature sets considered above. Feature binding sites reflect the time or the location at which a feature is realized for use (e.g., system compilation, installation, or execution). The feature binding sites correspond significantly to customer settings; varying them can yield unexpected insights about which features are relevant in which settings.

Another relevant technique (to the extent it has not been done systematically in the previous activity) is to use the setting characteristics (a kind of market profile) associated with each feature set in the DOMAIN MODEL to evaluate and flesh out the value chains. The setting characteristics can be used to scan the candidate customer settings for degree of matches to the profile.

Other, more formal techniques can be used to generate new value chains and correlate the features with customers in relatively systematic ways. Such techniques include:

- Transforming established value chains by projecting new links of the chain and looking for customers to fill the roles implied by those links; the transformation rules would be similar to some of the ones suggested in *Resolve Domain Model* but applied to "real" settings.

- Merging value chains into value networks involving a richer set of producer-consumer interactions among stakeholders. The risk here is that the more diverse the asset base, the more difficult the engineering problem, and if the target niches are too diverse there may be no economy of scope in treating it as a single asset base. For example, if there were such economies of scope, the asset base could be planned in terms of three distinct asset base customers (e.g., application developers applying Web, Motif, and ORB strategies) who would all target the same or similar end-user market segments.

The value chains and associated information already in the CUSTOMER DOSSIER should be revised to reflect the results of this analysis. In addition, it may be useful to record the correlations between features and customers/settings in a more direct form. The FEATURE/CUSTOMER MATRIX template in Exhibit C-13 illustrates one way in which this information can be captured.

> *Example.* Exhibit 81 shows an excerpt from a BUSINESS OPPORTUNITIES TABLE worksheet for the Outliner domain example. This worksheet shows two value chain scenarios. In the first, Webonautics is playing the role of Asset Producer, Asset Base Manager, and Application Developer to service the needs of two End-Users, HMS Blockhead and AFDSO. The table shows that there are two key feature sets involved in the scenario and explains the motivations and strategies in text beneath the stakeholders. In the End-User column, two "profiles" are identified that were derived from the DOMAIN MODEL and used to help match the feature sets with the End-Users (the profile descriptions are summarized beneath the table).

In the second scenario, Webonautics is again the application developer, focusing this time only on AFDSO as an End-User and using assets produced and managed by Persona.

Feature Sets	Stakeholder Settings			
	Asset Producer	Asset Base Manager	Application Developer	End User
• Structured information available in tabular (spreadsheet style) format. • Automatic translation of structured information from various PC and database tools to Web-publishable format supported.	Webonautics	Webonautics	Webonautics	HMS Blockhead, AFDSO Profiles 2 , 4
	Here, the end user customer wants the spreadsheet data to remain in its native format where it can be maintained and developed with the originating application and then be published on a need-to-know basis within the organization. As such an asset base of routines needs to be created (these can be little scripts) that, depending on the type of data, can simply generate formatted HTML OR call upon a helper application (e.g., Excel itself) to display data on behalf of the end user. Both of these may be required as not all users may have software on their PCs to act as helper applications.			
• Structured information can be displayed "graphically" (at least at a high enough level) with nodes and arcs rendered on a user's screen. • Switch between presentation styles (e.g. table vs. graph) controllable via user preference selection.	Persona	Persona	Webonautics	AFDSO Profile 9
	Here, Persona is trying to leverage expertise it presumably has in house and make that expertise available in a Web context. It may be that application work in this context (e.g., trying to support Webonautics end users) may lead to improvement in Persona's library of code to support tailored user interfaces and data presentation styles that it can use in its main stream UI work. This business context has two prongs, one relating to user selectability of display styles and the other for providing alternate display styles for the same data (e.g. table vs. outline vs. graph).			

Profiles:

...

2) External data needs to be accessed and displayed in structured format.

...

4) External data must be publishable in read-only format on the Web.

...

9) Information (both structure and content) that is gathered, changed and/or displayed needs controlled access

Exhibit 81. *Example Excerpt:* BUSINESS OPPORTUNITIES TABLE Worksheet

➤ Validate Results

Formally validate the feature-customer correlations in the CUSTOMER DOSSIER for accuracy, completeness and consistency. Update previous workproducts as necessary; for example, if new attributes are discovered that are useful to the setting characteristics/profiles in the DOMAIN MODEL, these should be handled by iterating back to the *Resolve Domain Model* task. Similar iteration can be done to update the DOMAIN STAKEHOLDER MODEL to include new customers that were not among the original stakeholders.

Empirical validation of the correlations in the CUSTOMER DOSSIER, which are still somewhat hypothetical, is performed as part of the next task, *Prioritize Features and Customers*.

When to Stop

- All core customers have been identified based on PROJECT OBJECTIVES and PROJECT CON-STRAINTS.

- The set of features and customers that will be analyzed in more detail has been filtered as much as possible to minimize the subsequent selection effort.

- Each feature set identified in the DOMAIN MODEL has been mapped against each candidate customer.

Guidelines

- <u>Treat stakeholders as candidate customers</u>. In the *Model Domain* phase it was important to treat stakeholders primarily as informants, even though many informants are potential customers as well. In the *Engineer Asset Base* phase, domain engineers need to treat stakeholders primarily as customers, even though their help as informants may still be required at various points in the process.

- <u>Do not try to meet the full spectrum of any specific customer's needs</u>. The asset base need not provide all system requirements for any potential customer setting; nor even all *domain* functionality required in these settings. The asset base must provide sufficient domain coverage to motivate application engineers to utilize assets in developing systems. Aspects of domain functionality may be left to application engineers to implement if a clean interface from asset-supported to developer-customized components can be provided.

7.1.2 Prioritize Features and Customers

In the *Resolve Domain Model* task, modelers described characteristics of potential settings where given features can apply. In the *Correlate Features and Customers* task, these potential market profiles for features were *grounded* in a set of specific candidate customers and settings. The resulting CUSTOMER DOSSIER provides an initial correlation of features to these customer settings. In order to make a strategic selection of the features to support and the customers to target for the asset base, the *Prioritize Features and Customers* task now analyzes the correlated features and customers in more detail and prioritizes them according to criteria such as value, utility, and feasibility.
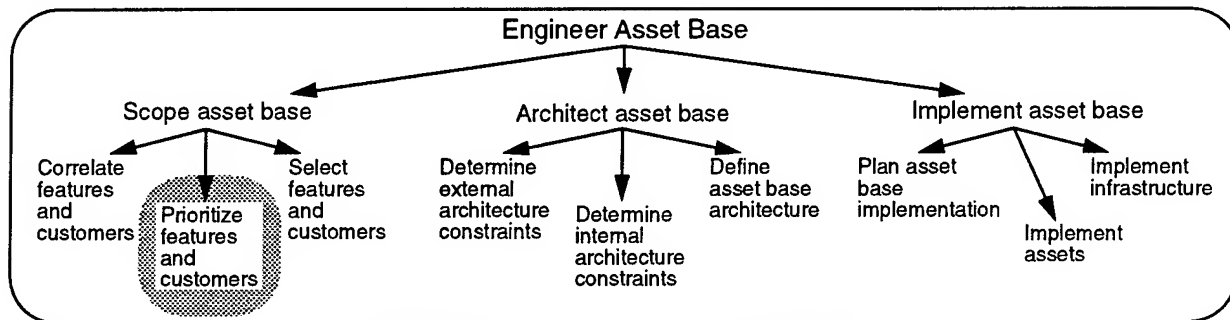
Exhibit 82. Prioritize Features and Customers Process Tree

The principal benefit of separating this task from its predecessor, *Correlate Features and Customers*, is that it helps to distinguish between activities focused on reestablishing the stakeholder context and activities focused on asset base design decisions. This methodological step helps mitigate

the dual risks of determining asset base requirements on the basis of one dominant customer or on the basis of no explicit customer at all.

The principal benefit of separating this task from its successor, *Select Features and Customers*, is that it helps to distinguish activities focused around evaluation and weighting from final selection. This allows for a certain amount of "what-if" analysis, forecasting, and validation with candidate customers before closing the decision process.

Approach

Among the issues addressed in this task are:

- How valuable would a given feature set be to a given customer? Would reuse be at a coarse or fine-grained level? Individual components or entire sub-systems?

- How many instances of use would be likely for each customer setting? How cost-effective would reuse be for the customer? How much would they pay?

- How feasible would it be to implement and maintain a given feature set?

A key challenge in this task is establishing a valid empirical basis for utility and feasibility assessments. Here, *utility* refers to the level of need or interest in a feature or feature set among the candidate customers; *feasibility* refers to the technical difficulty and cost associated with providing the feature set. Features, by their nature, are transformed significantly from concrete implementations; this means that there is no simple way to survey the potential number of instances of application. Similarly, it is difficult to assess feasibility without committing to or assuming a particular implementation technology or architectural approach.

Workproducts

■ ASSET BASE DOSSIER

The ASSET BASE DOSSIER includes the following information:

- A thorough characterization of each feature set and correlated customer settings. This dossier includes information in the CUSTOMER DOSSIER produced in *Correlate Features and Customers* and extends it with additional customer and feature attributes (including information from the DOMAIN DOSSIER) and information about feature utility and feasibility within each setting.

- A mapping between the feature sets and the customer settings to which they apply, prioritized in terms of key criteria such as utility and feasibility. This is a refinement of the initial mapping between features and customers produced in the previous task, *Correlate Features and Customers*. The FEATURE/CUSTOMER MATRIX worksheet template in Exhibit C-13 suggests a format for expressing this information.

Information in the DOMAIN DOSSIER about those representative systems studied during *Model Domain* that are relevant to the candidate customer settings should be carried over into the ASSET BASE DOSSIER.

When to Start

- An initial version of the CUSTOMER DOSSIER has been completed. This serves as a filter to

reduce the complexity of data to be analyzed.

Inputs

- CUSTOMER DOSSIER. Developed in the preceding task, *Correlate Features and Customers*, the information in this workproduct is extended in this task to incorporate more detailed data about utility and feasibility of various feature sets.

- DOMAIN DOSSIER. A source for information about legacy artifacts that may affect feasibility and/or level of effort to implement certain feature profiles; and constraints from customer settings that were analyzed as part of the *Model Domain* phase.

- CANDIDATE CUSTOMER INFORMATION. Supplemental information from potential customers about feature preferences and priorities, used for the trade-off analysis performed in this task. CANDIDATE CUSTOMER INFORMATION supports identification of features that must be available as a set and features that must be excluded from a set for the set to be usable to a specific customer.

Controls

- PROJECT OBJECTIVES. Needed to guide trade-off analysis; particularly to establish what degree of thoroughness is required, based on project expectations.

    Example. If the PROJECT OBJECTIVES specify that the project should demonstrate a factor of two reduction in development costs for application groups that become asset base customers, this might lead to prioritizing customers with potential for large-grained reuse over lower levels of reuse by more customers.

- PROJECT CONSTRAINTS. Useful for prioritizing customers and assessing feasibility of features.

Activities

In the *Prioritize Features and Customers* task, asset base modelers use the information about customers and features captured in the CUSTOMER DOSSIER as the starting point for a cost-benefit style analysis of different feature sets and different asset base customers and markets. Although the focus of this analysis is on the candidate asset base customers and their feature needs, each of these customers is considered in the context of the value chains in which they participate. Thus, the information in the CUSTOMER DOSSIER about value chains and their constituent settings plays an important role in this process. The final selection based on the prioritization in this task is made in the succeeding task, *Select Features and Customers*.

➤ Select analysis methods

Market analysis methods are an essential component of the *Prioritize Features and Customers* task. In the ODM context, this is the point in the life cycle where market analysis techniques are most directly applicable. In this task, asset base modelers assess the *demand* for various specific feature variants and overall system configurations within given *market segments*. See Section 8.4 for more details on Market Analysis supporting methods and their role in ODM.

The parallels between domain analysis and market analysis are striking enough that some researchers have described domain analysis as a whole as market analysis specialized to reusable

software. However, there are some important unique factors to be considered in applying market analysis techniques to domain engineering. These are:

1) Conventional market analysis is organized around *new product development.* An important aspect of the ODM approach to domain engineering is capitalizing on legacy artifacts and technology. This is a different process in many ways from product development *per se.*

2) Conventional market analysis is oriented toward *manufactured goods.* Domain engineering, at least in the scope of this Guidebook, is oriented around software or information reuse. Many assumptions about packaging, distribution, economies of manufacture, etc., that are implicit in conventional market analysis techniques simply do not work the same way in the reusable software marketplace. The high degree of variability obtainable in software is a related complicating factor.

3) The target market of an asset base is *application developers.* They are the immediate customers for domain engineering. This means that the domain engineer must consider at least two separate customer settings: the immediate (i.e., application developer) and the indirect (i.e., end-user), as well as possibly others (e.g., the asset manager). This introduces still other factors to be considered as part of the task.

The remaining activity sub-sections will describe elements essential to performing this analysis in the domain engineering context.

➤ Characterize candidate customer settings

Characterize candidate customer settings in the ASSET BASE DOSSIER to provide supplementary information to guide asset base scoping and asset implementation decisions. This may involve gathering additional information about settings already documented in the CUSTOMER DOSSIER or DOMAIN DOSSIER or studying new settings. These settings might now be studied to determine whether the features deemed of interest are in fact of interest to those practitioners.

While this may appear like a return to data acquisition, the process followed is quite different than that employed during the *Model Domain* phase. Here, the model of features has been created and the information-gathering is more in the nature of a focus group or market analysis. The focus is on validating the linkage between features and anticipated customer interests. The basic question being asked is: "If an asset satisfying this set of features were available, would you use this asset rather than implementing the capability from scratch?"

In some cases it may be determined that a given set of features requires augmentation with certain other features before an asset supporting those features would get used. Often there will be large sets of functions that must be provided as a whole; e.g., few people would use a single X-window routine out of a library. In other cases it is possible that a feature might need to be *excluded* from a feature set in order to gain acceptance. Thus, it may be useful to perform this analysis at the level of individual features rather than feature sets in order not to mask a customer's interest in a specific feature by hiding it within a feature set that is of less interest as a whole. Note, however, that this risks erosion of the integrity built into the feature sets during *Model Domain.*

> *Example.* In the Outliner domain, suppose that the DOMAIN MODEL defines a variant configuration that strips away editing functions, preserving only outline read-only browsing capabilities. Users of help systems for on-line applications have been identified as a candidate customer setting for this restricted variant of outlining capabilities, a class of application contexts not studied during the *Acquire Domain Information* sub-phase.

305

➤ Acquire supporting data

More information may be needed for a number of customer settings, even for representative systems already studied in the *Model Domain* phase. Consult CANDIDATE CUSTOMER INFORMATION from various sources, including the candidate customers themselves. Document any new information acquired in the ASSET BASE DOSSIER.

> *Example.* Assume that a number of requirements specifications, following a particular standard, were examined during the *Model Domain* phase. The DESCRIPTIVE MODELS may have noted certain features of these requirements documents, including their internal structure. Now, in prescriptive modeling, we must consider whether the asset base will itself contain requirements text and/or requirements documents (or document templates, fragments, generators, etc.). If so, we will probably require a finer level of detail about the use of such documents within the various customer settings.

➤ Quantify utility estimates

Prioritize feature profiles according to the estimated frequency and distribution of use of assets within the marketplace. Document the core or typical setting in which feature variants will most probably be used, as well as more peripheral opportunities.

There needs to be *sufficient* potential use for a feature variant to warrant its inclusion in the feature profile for an asset. One approach to estimating the "market share" of feature variants within the candidate customer settings might be the following:

• Across instances of use, come up with estimated scale of cost-to-adapt per instance of use;

• Characterize each variant across its instances of use using cost-to-adapt estimate;

• Allow for instances excluded or enabled by a given variant as well as straight comparison.

> *Caveat.* Estimating the potential market for a set of features within a set of potential customer settings is far from an exact science. Current methods and automated support for generating such quantitative estimates are limited. This is in part because reusable assets may employ generative as well as component-based techniques; thus potential application settings might not be readily apparent from code-based analysis.

➤ Analyze feasibility

Analyze feature combinations and architectural variants from the standpoint of feasibility. Consider the level of effort required both to implement the features and to maintain and do configuration management on separate variants within the asset base. Some feasibility issues to consider include the following:

• Functionality/performance trade-offs. Support of variability always carries a price, whether in performance, asset base overhead, complexity of the asset search and retrieval process, or increased complexity in configuration management and asset maintenance.

• Uncertainty in estimation. Cost of implementation for features implemented in previous systems should be more predictable than for novel features. If project metrics are available for exemplar system development efforts, levels of effort for implementation and maintenance of artifacts implementing desired features may provide some guidelines. Clearly, however, such estimates will be far from an exact science. Estimates will be even more difficult for novel feature variants. Additional input from domain experts and experienced asset developers will

probably be necessary.

- Technology assumptions. Although some working assumptions must be made here about the implementation technology to be used, the purpose here is not to make the technology choice, merely to gain additional information for the trade-off analysis of selecting features to be covered in the asset base. As much as possible, avoid preemptive commitment to particular implementation techniques. Feasibility assessments will need to be revisited after asset implementation choices have been made. Closely related variants may appear to necessitate independent and separately maintained code components; however, if generative techniques are to be used the variants may be derivable (and maintainable) from a single source work-product.

- Configuration Management. Consider configuration management (CM) issues as an aspect of maintainability. Providing separate variants can be compared to the CM problem of a system where multiple versions are simultaneously fielded and must be managed in parallel.

➤ Analyze utility and feasibility trade-offs

Synthesize the results of utility and feasibility analysis. Validate that distinct feature clusters have sufficient "breathing room" across the intended range of customers to warrant the overhead of separate creation and configuration management. This reflects a general rule in designing for reuse: the reusability of each asset must be considered within the overall context of the asset base within which it will reside, not just in isolation. (See Exhibit 83.)
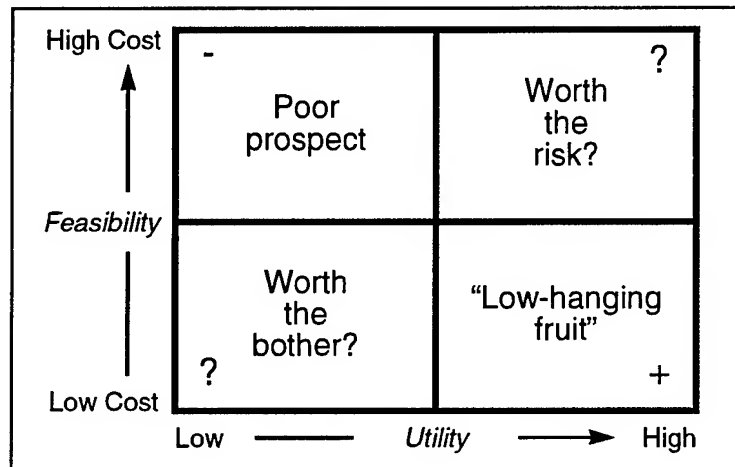
Exhibit 83. Utility-feasibility trade-offs

Some key trade-off issues are discussed below:

- Product line segmentation. One trade-off to be considered is analogous to that in product family packaging decisions: ensuring that variants supporting particular combinations of features have sufficient demand across the intended range of customers. Assess the anticipated flexibility of different customers with respect to feature profiles optimal for their needs. Also assess the tolerance of potential customers for small, incremental variations in assets. Supporting too many variants that differ in subtle ways irrelevant to users may actually weaken the utility of the asset base. Thus there is a inherent trade-off from the standpoint of utility. This discusses utility aspects of this trade-off; feasibility aspects are discussed below (e.g., the overhead of separate creation and configuration of closely related variants).

- Feature "distance" between variants. The value of a given asset within the asset base is deter-

307

mined in part by its *feature distance* from related assets. Formally, a distance measure can be defined for any two feature variants in terms of their relative position within a feature model. This measure can be extended to feature profiles, as some function of the distance measures of the constituent features. Since a feature profile can be treated as a specification for a potential asset, the concept of feature distance provides a somewhat formal way of characterizing distances between potential assets in terms of those feature profiles.

> *Example.* In the Outliner domain, consider an outlining program that allows up to 9 heading levels, versus one that allows 15 heading levels. The question is: for a developer who wanted 15 heading levels, how much value does that variant provide relative to the 9-level variant? This value might be worth supporting in the asset base; it might require a simple change of parameter, or may have more pervasive impact on issues like screen display, memory allocation, etc.

> By contrast, would it be worth separately providing a 9 versus a 10 level heading version? This difference probably would neither significantly change the level of interest for a particular customer, nor greatly expand the market of customers for the asset base.

- Customer flexibility. An asset need not satisfy an application engineer's entire set of requirements, even within the domain scope. It must satisfy enough of the requirements to motivate its use, and must result in a perceptible cost-effectiveness and/or productivity improvement for the engineer. Robust mechanisms for integrating the asset into applications can encourage some flexibility in requirements.

> *Example.* In the example above, could a given application tolerate fewer heading levels? For writers or idea processor users, this might be acceptable. For an application involving use of an outline interface to display directory structures, on the other hand, this limit might be unacceptable. In this example, would it be simple or complex for the engineer to add extensions to the asset to obtain the extra functionality needed? Conversely, what if there were a desire to restrict the number of available heading levels below the system maximum provided?

The results of this trade-off analysis is the prioritization of features and customers. Document these results in the ASSET BASE DOSSIER. The FEATURE/CUSTOMER MATRIX template in Exhibit C-13 illustrates one way in which this information can be captured. This information may be similar in format to the initial candidate feature and customer correlations documented in *Correlate Features and Customers*, but expanded here to include more attributes and analysis results, and perhaps re-ordered to reflect prioritization decisions.

➤ Validate analysis results

Check estimates for internal consistency (allowing for the non-linearities discussed in the Guidelines below). If similar customer settings produce wildly different estimates, look for the explanatory factors and allow for error in the estimation process.

Obtain feedback and review from prospective customers. Provide them with specifications of individual assets or overall system configurations derivable from proposed feature variations. Their reactions will serve as early validation as to whether the feature combinations and system variants supported are deemed useful by application engineers.

When to Stop

- Sufficient supplementary data has been acquired about customer settings and feasibility issues.

- Estimates are documented of the utility and feasibility factors for each feature set with respect to each customer setting.

Guidelines

- Utility profiles need not be linear. More is not always better. In mapping feature sets to given customer settings, increased functionality will typically reach a peak, then decrease rapidly in terms of perceived utility.

- Minimize implementation assumptions. Note the constraints or caveats potential utilizers mention about the required form, structure, or other characteristics of assets. Where possible, though, suppress details about how assets will be implemented that might influence customers' assumptions about the usefulness of a given variant.

7.1.3 Select Features and Customers

The first two tasks in the *Scope Domain* sub-phase produced a great deal of information about the potential value of various feature sets to various candidate customers. The primary purpose of this task is to make the final selection of features and customers for the ASSET BASE; these decisions are recorded in the ASSET BASE MODEL.
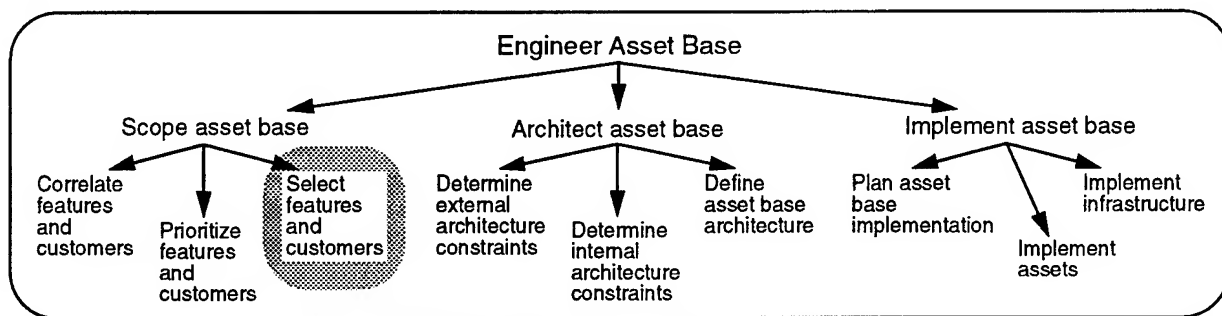
Exhibit 84. Select Features and Customers Process Tree

This task represents the fulfillment of one of the primary objectives of ODM, which is to facilitate the creation of specifications of reusable assets that make clear not only their functionality (via the feature profile) but their *intended scope of applicability* (via the customer-related information in the ASSET BASE MODEL). By selecting the features and customers and documenting the linkage, the rationale for each is integrated into the model. This rationale provides valuable information downstream for the implementor of the ASSET BASE INFRASTRUCTURE, who will need to support potential utilizers searching for assets. They may be best able to characterize what they need in terms of their own customer profile, which can be mapped to the feature profile using, in part, information generated in this task.

Approach

There are several important aspects to this task in relation to the overall process model, including:

- Features and customers are selected in parallel, rather than in a sequence driven by one factor over the other. Maintaining this balance between the two, while a difficult decision procedure, helps ensure the overall quality of the asset base; it makes it more likely that there will be a coherent market of customers specified for the asset base, as well as a coherent range of features supported.

- Decisions on features to be supported precede architectural decisions about how to support these features in the asset base.

Note that the latter is a point of possible confusion, because the features themselves may contain requirements for supporting particular *system architectures* in the assets and the applications to be built using assets. However, what is *not* decided within this task is how the asset base will deliver components in the proper configuration.

> *Example.* In the Outliner domain, the ASSET BASE MODEL may specify that two versions of the outliner asset must be supported, one that is cleanly separated from the host word processor program, communicating with an event-driven interface; the other, a system that can be tightly integrated with specific word processors based on some customization/reconfiguration process. Each of these represents decisions roughly correlating to *system architecture* decisions. The *asset base architecture* decisions still deferred would include issues such as: how much sharing between code components will take place between the two versions of the program?

Workproducts

■ ASSET BASE MODEL

The ASSET BASE MODEL includes the following information:

- The set of features to be supported by the ASSET BASE, as well as relationships and constraints among the features.

- The set of asset base customers, characterized in terms of attributes related to the settings in which the selected features will be applicable (e.g., role in life cycle phases, experience level, etc.).

- A mapping of selected features to selected customers and settings. This is a refinement of the mapping between prioritized features and customers in the ASSET BASE DOSSIER produced in the *Prioritize Features and Customers* task. The FEATURE/CUSTOMER MATRIX worksheet template in Exhibit C-13 offers a format for expressing this information.

■ ASSETS OF INTEREST

The set of assets seen at this time (somewhat speculatively) to be desirable end products of the *Engineer Asset Base* phase, based on the feature and customer selection.

When to Start

- Candidate customer settings for the ASSET BASE have been identified and mapped to the DOMAIN MODEL in *Correlate Features and Customers*.

  Deciding on features and customers for the asset base without first performing this task means that any assumed connections between customers identified and features to be supported are implicit only. There is no way to validate and verify the connections, and therefore no way of checking whether viable choices have been made.

- The correlations have been refined, and to some degree prioritized, by assessing the potential utility and feasibility of each feature set, relative to identified customers.

  Deciding on features and customers for the asset base without first performing this task

means that the decisions may be skewed by the minimal correlation first produced. So, for example, a feature set of interest to several customers might be preferred to one of interest to just one customer, when in fact the potential instances of use for the former are much fewer than for the latter.

Inputs

- DOMAIN MODEL. Provides the starting point for the ASSET BASE MODEL. Also includes information to supplement the prioritizations in the ASSET BASE DOSSIER and further support the selection process.

Controls

- ASSET BASE DOSSIER. Information about the features and customers to be selected in this task, as prioritized in *Prioritize Features and Customers*. This serves as both a control for this task (via the prioritizations that guide selection) and an input that contributes information to the ASSET BASE MODEL.

- PROJECT CONSTRAINTS. Estimated level of effort for implementing various feature sets are mapped to constraints such as budget, schedule, available staff and resources, etc. By deferring consideration of these constraints to this task, information in the ASSET BASE DOSSIER produced in the previous task (as exemplified by the FEATURE/CUSTOMER MATRIX template) can be reused if project resources expand.

Activities

➤ Make selection and document

Based on PROJECT CONSTRAINTS, the prioritizations in the ASSET BASE DOSSIER, and any other information or insights that can be obtained about the domain engineering organization and the candidate customers/settings, make the final decisions about which features the asset base will support in which customer settings. Document these decisions in the ASSET BASE MODEL. This model represents the binding commitments or specification for the overall feature capabilities of assets within the asset base. It should be a subset of the features in the DOMAIN MODEL. It can be represented in various ways, such as a restriction or pruning of the DOMAIN MODEL. The ASSET BASE MODEL is termed a model because it should retain all semantic information relevant to selected features. However, the model is created through strategic transformation and filtering of the previous model, *not* by an analogous process.

Consider the selection process in terms of one feature. One option is simply to select a single variant as the variant to be supported in the asset base. Other variants are excluded from the ASSET BASE MODEL; asset engineers need not address how to obtain them. Another option is to implement both variants, even if both could not co-exist in a single run-time system. The asset implementor might develop both variants and store them in the asset base, or implement a generator that can create both variants as instances, to be generated on request by asset utilizers or generated and archived. Some features or setting characteristics within the original descriptive scope of the domain model may be excluded from the ASSET BASE MODEL. Typically, the reason is insufficient need for that functionality in the anticipated customer settings.

Where architectural variants have been rejected as part of the ongoing asset base, consider these variants as incremental builds in a phased development plan for the asset base. This information will feed forward into the detailed planning activities of the *Plan Asset Base Implementation* task.

➤ Integrate and validate data

Since definition of the ASSET BASE MODEL involves both top-down and bottom-up processes, some integration and cross-validation will be required, particularly if the activities have been performed in parallel. Specifications of individual feature variants, feature configurations, and architectural variants to be supported should be integrated and checked for both internal and cross-consistency.

➤ Identify assets of interest

Identify a preliminary set of ASSETS OF INTEREST that are seen likely at this time to be desirable end results of the *Engineer Asset Base* phase. Assets are discrete system/software entities that encompass some set of feature variants, so the objective here is to assemble the selected feature variants into groups that will be engineered in concert as individual assets. This determination, somewhat speculative at this point, takes into account historical feature clustering, new feature clusters that were discovered as part of the feature-customer analysis, selected feature binding sites, and so on.

The ASSETS OF INTEREST provide a starting point for the more detailed analysis to produce the ASSET BASE ARCHITECTURE and ASSET SPECIFICATIONS in the ensuing *Architect Asset Base* subphase.

Consider the following forms that assets can take:

*   ***Engineering assets***. Engineering assets include the traditional notion of reusable components (possibly spanning the engineering life cycle) that will be retrieved by application engineers, possibly tailored or modified, and then incorporated into their own products. Some comparative sampling of engineering artifacts of various types should have been done as part of the *Acquire Domain Information* sub-phase of the *Model Domain* phase. This information, available in the DOMAIN DOSSIER, can help asset developers identify candidate artifacts for reengineering.

    Other artifacts that are candidates for transformation into assets are interim artifacts used in system development but not part of the deliverable system. Examples include test data, debugging scripts, requirements checklists used by analysts as reminders, etc.

*   ***Process assets***. Encoding, formalization, or automation of a work process in the domain, typically used as a tool, decision aid, etc., rather than a reusable component to be directly incorporated into system artifacts. Potential process assets mainly emerge in the *Interpret Domain Model* task within the *Refine Domain Model* sub-phase of *Model Domain*. In the *Select Features and Customers* task the surrounding context and rationale for these process assets is investigated. Often, the result can be discovery of potential types of assets never envisioned at the start of the domain engineering project.

*   ***Learning assets***. Closely related to process assets, learning assets are structured around the roles, profiles, or experience of particular stakeholders. Since a given stakeholder may utilize multiple workproducts and perform many processes, opportunities for encapsulating reusable knowledge into assets may emerge independently of artifact or process structure. For example, automated or codified expert knowledge could be used in decision making or workproduct transformation. Some of the products of domain engineering (like the DOMAIN MODEL itself) can be considered learning assets.

➤ Allocate selected features to customer settings

Allocate features to each customer setting within the scope of the asset base. For example, a feature that affects the implementation language for a component would be allocated to a development setting; a feature that specifies how the component interacts with an end-user would be allocated to a usage setting. The FEATURE/CUSTOMER MATRIX template in Exhibit C-13 illustrates one way in which this information can be captured. This information may be similar in format to the prioritized candidate features and customers, but trimmed here to include entries only for the selected features and customers.

➤ Contract with customers for commitment

This is in large part a validation step, but an essential one at this point. The asset base is going to be designed based on a set of assumed customers. Before proceeding further, some contracting with these customers must be done.

When to Stop

- Core customers' needs have been addressed in the prescriptive features selected for the ASSET BASE MODEL.

- The features selected for the ASSET BASE correspond to one of the layers identified in the DOMAIN MODEL. This ensures that the features supported will not have gaps or excess coverage that are responsive only to the selected customers.

- The features selected in the ASSET BASE MODEL are feasible given the project constraints.

Guidelines

- Supplement written priorities with business intuition and insight. The business opportunities, as identified in *Correlate Features and Customers* and further elaborated and evaluated in *Prioritize Features and Customers*, help to determine which features are going to be most useful/feasible/profitable in the asset base. This data alone probably will not be sufficient in many cases; it will need to be supplemented with other less formalizable business and technical factors unique to the organization, requiring business intuition and skills for interpreting and dealing with localized political and economic constraints.

- Consider resource trade-offs. The final ASSET BASE MODEL can be optimized in at least three ways:

   — Satisfy core customers with the minimal set of features that will obtain their commitment as users of the asset base; beyond this point, use minimum project resources;

   — For the allocated resources, satisfy the core customers with the richest set of features of interest to them;

   — For the allocated resources, satisfy the broadest market of customers possible.

- Consider trade-offs between optimization and generality. One primary trade-off will be between closeness of fit to particular exemplars (at this point, customer settings) versus the flexibility to be used by multiple customers. A priority to integrate with one existing customer system will create strong pressure to adopt an architectural approach compatible with that exemplar system. To avoid undue influence of one existing architecture make sure you have studied several as diverse as called for by the domain definition.

313

7.2 Architect Asset Base

The *Scope Asset Base* sub-phase has produced an ASSET BASE MODEL that specifies the range of features to be supported by the ASSET BASE and the targeted customers for the assets. This model even captures to some extent which *combinations* of features would be desired within particular application contexts, e.g., by specifying the desired feature binding sites for selected features.
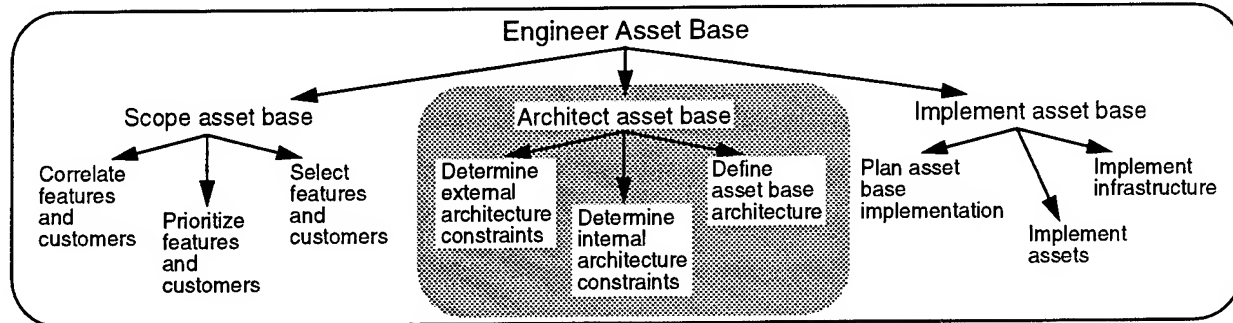
Exhibit 85. Architect Asset Base Process Tree

However, as yet the selected features have not been allocated to specifications for individual assets. There are many potential axes of variability for implementing assets that support these feature profiles, including:

- Whether desired functionality would be encapsulated for customers in monolithic components or aggregates of smaller components;

- How much internal reuse will occur within assets, i.e., how strongly interconnected the asset base as a whole will be;

- Whether alternative functionality required by different customers will be implemented as separate variants, or generated/transformed as required.

The primary purpose of the *Architect Asset Base* sub-phase is to isolate a subset of these design decisions at the level of the ASSET BASE ARCHITECTURE. Architectural concerns include determining to what extent the design and implementation of assets is constrained by the external interfaces anticipated in different customer settings, as well as determining the internal structure and interconnections of components in the asset base.

Approach

The ODM process to this point includes steps that are needed to address the problems of interpreting and transforming legacy artifacts into assets with predictable scope of applicability. The process is capable of supporting the construction of *individual assets*, implemented using a spectrum of component-based or generative techniques, and documented via a feature language attuned to the domain-specific terminology and application interests of practitioners.

It is clear that the process as described demands a non-trivial investment of resources in comparison to single-system engineering. The **architecture-centric domain-specific** view of reuse is based on the claim that long-term economies of scale and scope to justify the added investment will be best achieved by building asset bases focused around highly specific application areas, and organized into coherent architectures enabling large-scale reuse of aggregates of individual components. A central problem in this approach to reuse is the design trade-off between supporting large-grained reuse by application developers (which requires *structural integration* of compo-

nents) and supporting *high variability* in structure to enable reuse within a broad market of applications.

Besides supporting the economics of reuse, an architectural approach to implementation of asset bases addresses a problem actually introduced by the formal ODM feature modeling approach. Fine-grained modeling of feature variants, choice of generative and component techniques, and the examination of diverse customer settings all combine to create a potential combinatorial explosion in variability. More specifically, these techniques *expand modelers' abilities to envision potential variability* for their products, and thereby increase their need for techniques to deal with it effectively.

Reducing variability through architectural "chunking" into manageable sets of features is one important technique. However, this strategy tends to restrict the range of variability that can be supported and must be traded off against the variability needs of targeted asset base customers. Support for variability at the architectural level is one of the current areas of research in the field of software architecture [Clem95a]. A key ODM theme is the attempt to systematize domain engineering in a way that facilitates the specification and implementation of *highly variable architectures*.

As illustrated in Exhibit 86, the way domain boundaries are drawn with respect to their exemplar systems has a dramatic effect on the requirements for support of architectural variability within those domains. The grid represents a range of possibilities shaped by two dimensions. *Historical relatedness* refers to the "genealogical" relations among systems: whether the systems were developed, maintained, used by the same people in the same settings. *Structural similarity* refers to the topological closeness of systems, in terms of their component structure and interconnections. Historical relatedness can be viewed as closeness among systems from a *problem space* perspective, while structural similarity addresses closeness from a *solution space* perspective.
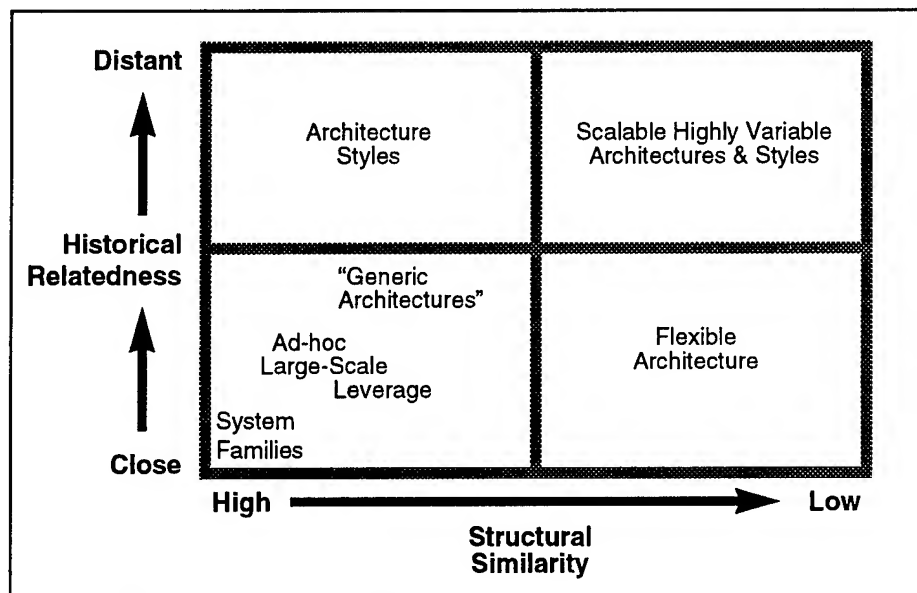
Exhibit 86. The Variability Barrier in Software Architecture

Although most examples that come readily to mind suggest close correlation between these dimensions, they are independent. System families in the conventional sense reflect lineages (as the term "family" implies), usually built by the same organizations (perhaps the same people) and often delivered over time to the same customer or market. But not all such lineages exhibit struc-

tural compatibility from system to system; this is a characteristic that varies from domain to domain. As we move away from system families with high structural similarity, we encounter the need for *flexible architectures*. Since the current state of the art does not readily support the systematic construction of systems along these lines, tractability usually depends on *ad hoc* techniques and a high degree of expertise on the part of developers, supported by ample knowledge of the application context.

Conversely, researchers in software architectures have recognized dramatic structural similarities in the architectures of systems built to perform diverse real-world missions with little or no historical relationships, developed and used by diverse communities. Some of these structural patterns are beginning to be codified and catalogued as *architectural styles* [Shaw96].

As the diagram in Exhibit 86 suggests, current architectural practices can succeed to the degree that they can exploit either high structural similarity or historical relatedness. But these solution-space and problem-space perspectives do not directly correlate in general; and in many practical business settings, *problem domains* are the motivating charter behind domain engineering initiatives.

This presents a formidable challenge for domain engineering methodology. At a minimum, it would be desirable to at least be able to identify and avoid domains where current architectural methods do not suffice. A more ambitious goal is to develop a repertoire of techniques that can address the "variability barrier:" scalable, highly variable architectures and architecture styles, explicitly embedding enough contextual information to support reuse in diverse application areas and organization contexts.

ODM offers significant progress towards these goals, but does not provide complete solutions for these long-ranging research problems. However, the core elements of the ODM process are all *necessary*, if not sufficient, to achieving reuse at this scale. The task structure in this phase is designed to be scalable to asset bases architected to support high variability within diverse settings. There is, however, a strong dependence here on supporting methods in software architecture and related areas, and advances in these supporting methods will be required to fully achieve the ODM engineering objectives.

The approach presented here involves clearly specifying the potentially numerous constraints on variability among application architectures derivable from the asset base, and then resolving these multiple constraints to shape the architecture of the asset base.

Constraints on variability in the feature space have been introduced throughout the domain engineering life cycle. The constraints to consider at this point in the process are drawn from a variety of sources, such as the prescribed asset base feature sets, the characteristics of the customer settings and specific target applications within those settings, resource and technology limitations within the asset developer or customer organizations, and so on. These constraints are organized into two major classes, addressing variability at the architectural level in terms of:

- *External architecture constraints* imposed by those portions of potential target applications (and their surrounding environments) that lie outside the domain. These constraints impact the variability of the interfaces to and from the external functionality which the asset base must accommodate.

- *Internal architecture constraints* that reflect feature interrelationships within the domain. The fact that subsets of assets may be used as distinct collections in different customer settings introduces a dramatic increase in the number of potential variant structures to be considered. This is not the case in engineering for a single system, where the structure has impact primarily on maintainability and related engineering factors.

The ASSET BASE ARCHITECTURE that results from resolving these constraints must capture the functional and structural variability within an asset base and must support resolution of that variability to produce specific instances of domain functionality that can operate within target applications.

To resolve the variability, the ASSET BASE ARCHITECTURE must be able to map the features selected for each specific target system within the scope of the asset base to the particular configuration of assets that will supply those features. In other words, the ASSET BASE ARCHITECTURE must contain the information needed to determine how to instantiate and configure sets of assets to satisfy all the feature combinations that are allowable within the asset base. This information can be substantially more complicated to represent than an individual system architecture, which need only describe the characteristics of a single system (i.e., a single variant within the asset base scope).

In addition, an asset base generally includes assets that do not contribute directly to executable systems. These can include various kinds of documents, requirements specifications, test suites, and so on. A structural notion analogous to software architecture (i.e., the "architecture" of a requirements specification or a test suite) is relevant here, although the focus is not on system implementations. The ASSET BASE ARCHITECTURE must include the information needed to produce system-specific instances of these assets as well.

Note that the increasingly conventional notion of a *generic architecture* (i.e., a system architecture which generally has a fixed topology but supports component plug-and-play relative to a fixed or perhaps somewhat variable set of interfaces) does not satisfy the needs of an ODM asset base architecture in general, although it may suffice in some circumstances. Generic architectures, given the current state of practice, do not readily support the degree of variability that is often

Exhibit 87. Architect Asset Base IDEF$_0$ Diagram

317
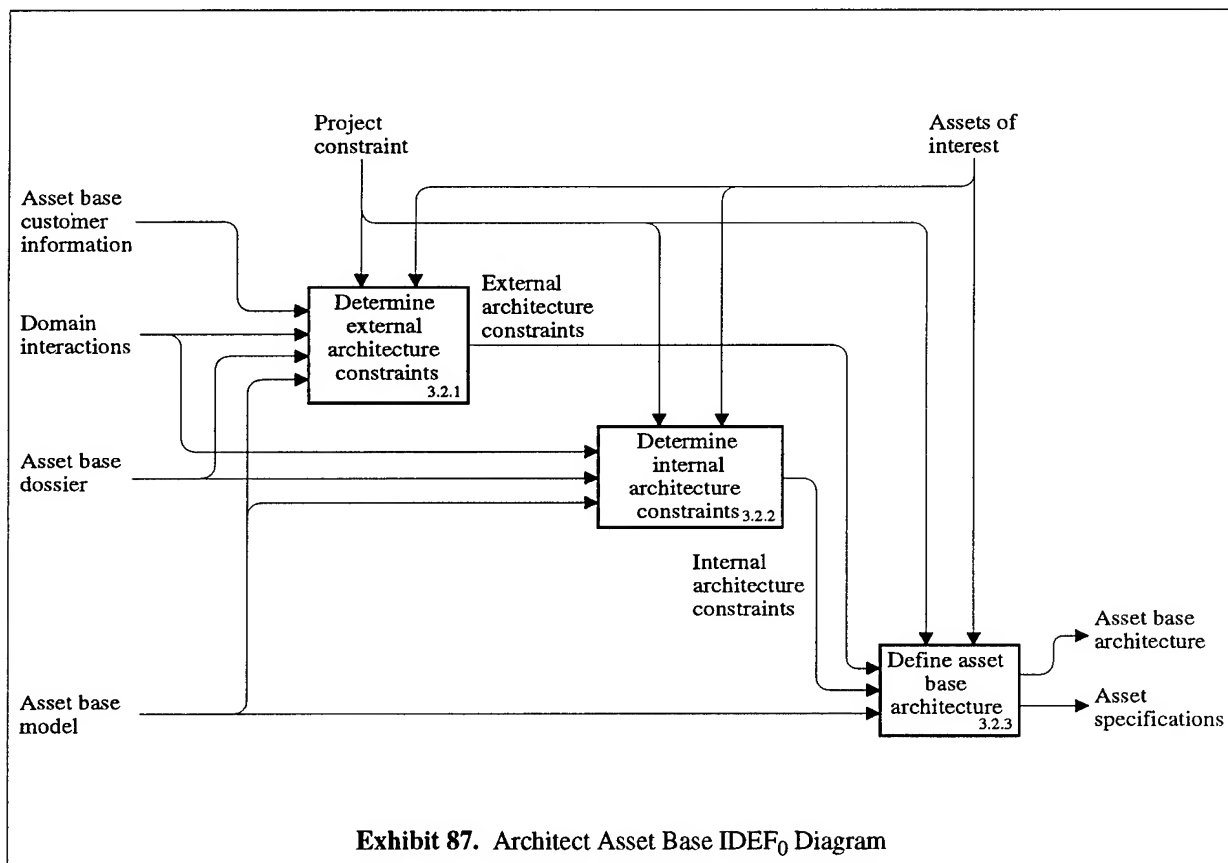
needed to satisfy diverse customer settings (e.g., topological variations, robust architectural sub-setting, generic subsystem-level architectures adapting to diverse system architectures).

It is important to note that not all domains require richly architectural solutions. Vertical domains, encompassing large portions of an application, generally will require architected solutions. Horizontal domains, however, may tend towards more taxonomic and less component and interconnection structure in the asset base structure.

> *Example*. Consider a domain engineering project focused on computer font data, including a taxonomy of typeface styles and operations upon fonts (e.g., condense, italicize, etc.). This is clearly a domain where useful results could be obtained without addressing conventional system architectural concerns.

Results

The primary results of this sub-phase are:

- An ASSET BASE ARCHITECTURE that models the range of variability among system architectures or configurations that can be derived from feature combinations in the asset base. It includes the rules, relationships, and constraints among assets that govern how they can be combined and instantiated to produce application artifacts supporting specific feature ensembles.

- A set of ASSET SPECIFICATIONS defining the required features, interfaces, and behaviors of each ASSET for implementation purposes. Each of these specifications is subject to an independent technology selection process in the next sub-phase, *Implement Asset Base*.

The ASSET BASE ARCHITECTURE and ASSET SPECIFICATIONS together define the requirements and constraints needed by the next sub-phase, *Implement Asset Base*, to build the assets and the infrastructure to support their use.

Process

As shown in Exhibit 87, the *Architect Asset Base* sub-phase consists of three major tasks:

- The *Define External Architecture Constraints* task addresses external interfaces to asset functionality required by each of the selected customer settings specified in the ASSET BASE MODEL. These include constraints imposed by the external system environments in which assets or collections of assets will operate, as well as external services that may be invoked by asset functions. In practice, there may be some overlap between these two kinds of external constraints, which necessitates their being dealt with in an integrated way. The common thread is that these are constraints over which asset base engineers have *no direct control* (other than to exclude consideration of a given customer setting by iteratively adjusting the ASSET BASE MODEL).

- The *Define Internal Architecture Constraints* task addresses the issues of how to structure the internal relationships between assets in the asset base as a whole, and in particular among collections of assets that will be used in concert in certain customer settings. In this task there are also two complementary perspectives considered: layering of assets in *separately selectable* subsets, each of which represents a restricted version of overall functionality; and *component* encapsulations of particular domain functions. Constraints on architecture are assessed with respect to these two axes of variability.

- The *Define Asset Base Architecture* task integrates the constraints identified from the first two

318

tasks and performs the final trade-off analysis that results in the ASSET BASE ARCHITECTURE. Committed features are allocated to individual ASSET SPECIFICATIONS. Each specification represents a partitioned area of the asset base that may be implemented with separate components, generative techniques, or hybrid strategies. In the *Implement Asset Base* sub-phase, these specifications will be mapped to specific implementation strategies. The architecture also specifies high-level interactions among assets.

Guidelines

- Small domains support high variability most effectively. Domains defined at the sub-system level have a better chance of scaling to highly variable architectural approaches. In general, the more variability required, the smaller the domain scope should be drawn. So, for example, it would be feasible to address a large-scale family of applications with a domain engineering approach, but only if a relatively inflexible architecture (e.g., a standard architectural framework or, at most, a "generic" architecture) were specified. In many organizational settings, however, there are reasons why such an architectural approach will not be adopted. Tackling a large-scale domain in such an environment is therefore a high-risk strategy.

- The *Determine External* and *Determine Internal Architecture Constraints* tasks can be performed sequentially, in parallel, or iteratively. For a single-system engineering context, it would make most sense to determine external constraints first, and use these to filter the options for internal structuring decisions. While this sequence also makes intuitive sense in the domain engineering context, there is the added complexity that external constraints reflect an intended multi-system scope of applicability determined (to some extent "designed") by the asset base engineers. This means that in principle opportunities or constraints emerging from internal architectural issues could influence external decisions such as the range of external facilities to address or the specific interfacing mechanisms supported.

- Defer resolution of conflicts among external and internal constraints to *Define Asset Base Architecture*. Managing the potential circularity in this process is part of the rationale for separating the constraint determination tasks from the *Define Asset Base Architecture* task, where binding decisions are made. Even though there may be circular dependencies in the various constraints identified in the first two tasks, the activities of documenting these constraints can terminate without trying to resolve conflicts in the same pass. The final task in the sub-phase can then apply decision techniques at a more or less formal level to produce an acceptable set of architectural choices.

7.2.1 Determine External Architecture Constraints

In an industry where domain engineering methods are mature and pervasive, many of the external interfaces identified would be interfaces to other asset bases or potential asset bases. However, for the foreseeable future domain engineering projects will be "islands of reuse" in an ocean of legacy systems. So identifying the smallest range of variability at external interfaces is an important scoping technique.

By determining these constraints here, the ASSET BASE ARCHITECTURE can be tuned to those constraints relevant to the selected customers.
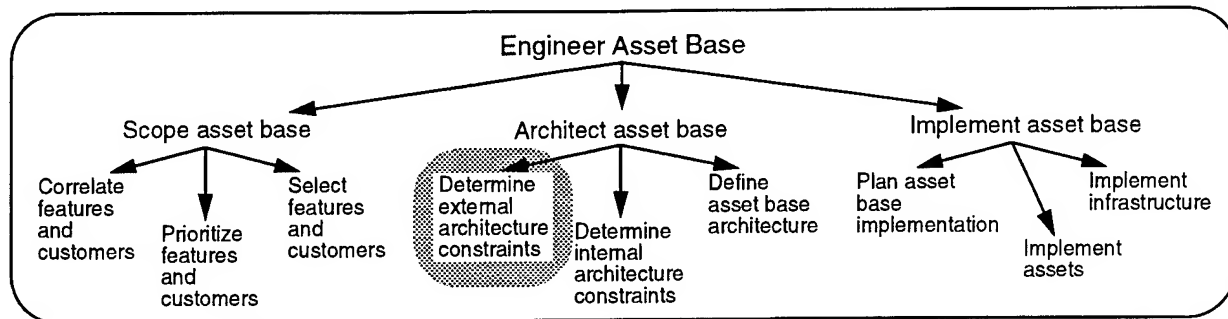
Exhibit 88. Determine External Architecture Constraints Process Tree

Approach

In this task, the focus is on identifying relevant external interfaces in the DOMAIN INTERACTIONS, constraining them based on the features selected in the ASSET BASE MODEL, and then further constraining them bases on the specific characteristics of selected customer settings.

> *Example.* In the Outliner domain, an example of an external architecture constraint might be a given customer's requirement that the outliner interface with a particular word processor program, which has its own internal architecture and protocol for communication with other programs. In addition, because the project is in part a technology evaluation/demonstration project, the outline assets must be implemented using an in-house generative tool—an example of a technology constraint on the project.

Workproducts

■ EXTERNAL ARCHITECTURE CONSTRAINTS

This workproduct includes the following information:

- A list of the key external interfaces to (and from) ASSET BASE functionality which the ASSET BASE must support, for each customer setting for which reusable ASSETS are to be developed. Interfaces include environmental and sub-system interfaces. Environmental and sub-system interfaces can be partitioned if appropriate, but it is possible that the determination of which category an interface falls into may depend on final architectural decisions that will be decided in a later task.

- An allocation of features from the ASSET BASE MODEL to the external interfaces relevant to each customer setting. The FEATURE/EXTERNAL INTERFACE MATRIX worksheet template in Exhibit C-14 illustrates one way in which this information can be expressed.

- Documentation of the relevant constraints each customer in the ASSET BASE MODEL has on each external interface. The CUSTOMER/EXTERNAL INTERFACE MATRIX worksheet template in Exhibit C-15 illustrates one way in which this information can be expressed.

When to Start

- The ASSET BASE MODEL has been specified in the *Scope Asset Base* sub-phase.

- Information about selected customer settings has been documented in the ASSET BASE DOSSIER.

320

Inputs

- <u>DOMAIN INTERACTIONS</u>. Structurally related domains reveal key interfaces with trade-offs such as strong versus weak coupling.

- <u>ASSET BASE MODEL</u>. Source of feature sets to be implemented and associated customer settings.

- <u>ASSET BASE DOSSIER</u>. Information about representative systems studied in the *Model Domain* phase and customer settings studied in the *Scope Asset Base* sub-phase. Information captured during *Prioritize Features and Customers* provides information for determining the constraints relevant in each customer setting.

- <u>ASSET BASE CUSTOMER INFORMATION</u>: A component data flow of CANDIDATE CUSTOMER INFORMATION, this includes new information elicited from selected customers about architectural constraints of systems targeted for implementing or migrating domain assets.

Controls

- <u>ASSETS OF INTEREST</u>. Initial candidate partitionings of features into assets, based on the feature-customer analysis in *Scope Asset Base*. These may be useful in allocating features to external interfaces.

- <u>PROJECT CONSTRAINTS</u>: Constraints on the domain engineering project that can impact the external interfaces. These constraints can include:

  — Constraints on *project resources*, such as budget and schedule, as well as restrictions on specific resources available for particular architectural options.

  — Constraints on *available technology*, including restrictions on strategies for architecting the asset base.

  These are distinct from the architectural constraints to be elicited from ASSET BASE CUSTOMER INFORMATION which concern constraints on system architectures in which assets will be integrated. The constraints discussed here directly affect the architectural choices of the asset base implementors.

Activities

➤ <u>Identify key external interfaces</u>

Starting from the development and usage settings and DOMAIN INTERACTIONS documented in the DOMAIN DEFINITION, identify key external interfaces to the asset base functionality to be implemented. Document these as part of the EXTERNAL ARCHITECTURE CONSTRAINTS.

Note that what is defined as an external interface at this point in the life cycle may differ from the definition during the *Model Domain* phase. A related area of functionality is now considered external if it is outside the scope of what will be implemented for the ASSET BASE. Since the ASSET BASE MODEL is a subset of the DOMAIN MODEL, some functional areas within the DOMAIN MODEL scope will appear as external interfaces to the ASSET BASE MODEL. Other areas within the DOMAIN MODEL scope may not need to be modeled as interfaces at all.

Example. In the Outliner domain, suppose that the DOMAIN MODEL includes the functional area of generating outline-style numerals for outline documents as one feature area. How-

ever, in the *Scope Asset Base* sub-phase it is determined that the primary customers targeted for the initial release of the asset base are users who rarely use this feature. When required it often calls for sophisticated typographic processing that is deemed outside the core functional area of the domain. This component would now be treated as an *external* interface. The external architectural problem now becomes finding a way to allow application builders to code their own routine of this kind and link it if needed with the core outlining capability.

On the other hand, if the ASSET BASE MODEL excludes outlines dealing with graphical displays of outline structure, then an interface with graphic display routines may not be required as part of the ASSET BASE ARCHITECTURE.

➤ Partition prescribed features to key external interfaces

Determine the features within the ASSET BASE MODEL that constrain each interface, and allocate the features accordingly. Use the ASSETS OF INTEREST to help guide this, as appropriate. This refines the features-to-settings allocation in the ASSET BASE MODEL to a greater level of detail in both features (e.g., individual features and feature variants within the feature set) and interfaces relevant to each setting. Document the results as part of the EXTERNAL ARCHITECTURE CONSTRAINTS (for example, in a format such as the FEATURE/EXTERNAL INTERFACE MATRIX worksheet template shown in Exhibit C-14).

➤ Determine constraints customer settings impose on each interface

For each customer setting in the ASSET BASE MODEL, document constraints they may impose on each external interface. Architectural constraints of legacy systems that are potential settings for asset base utilization should be considered as well as requirements for new systems. Much of this information should be available in the CUSTOMER DOSSIER. Supplemental ASSET BASE CUSTOMER INFORMATION is obtained through direct contact with customers. Document the results as part of the EXTERNAL ARCHITECTURE CONSTRAINTS (for example, in a format such as the CUSTOMER/ EXTERNAL INTERFACE MATRIX worksheet template shown in Exhibit C-15).

> *Example.* In the Outliner domain, for a stand-alone outlining utility that requires access to multiple text editing systems, a loosely-coupled interface might be most efficient. For an outliner embedded in a larger word processor, seamless and tightly integrated use of the host editor might be required. To build a set of assets that can be used to configure *both* versions, several variants might need to be implemented and separately supported. To implement the asset base, though, engineers first need to determine the *specific* text editor interfaces the assets will need to accommodate.

➤ Validate external constraints

- Validate for completeness: Ensure that the documented constraints are complete relative to the structural relations in the DOMAIN INTERACTIONS workproduct.

- Validate for consistency: Where the representations used encourage redundancy, make sure the data is consistent, particularly if different modelers have performed activities separately. For example, if two customer settings impose the same architectural constraint, the estimate of effort required to meet that constraint should be the same in each case.

- Validate via customer feedback: Obtain direct feedback about the constraints from individuals within customer organizations. Make sure that information in the ASSET BASE DOSSIER that may have been carried over from the DOMAIN DOSSIER is still current and accurate.

When to Stop

- <u>All key external interfaces have been identified</u>, and the list has been filtered according to relevant constraints. This ensures that all the parameters to be considered in the trade-off analysis have been collected, but that no analysis on easily excluded options will be performed.

- <u>All relevant constraints have been documented</u> for each customer. There has been no attempt to screen out certain sets of constraints to achieve compatibility. In general the set of constraints identified will *not* be mutually compatible.

Guidelines

- <u>Consider interface coupling trade-offs</u>. Most guidelines for designing reusable software advise developers to "minimize context dependencies" or encapsulate components as a general design rule. But design for reuse is harder than conventional system design in large part because each of these design decisions involve trade-offs, the impact of which must be assessed for multiple application contexts.

  Where possible, encapsulate asset base functionality for easy integration with other software resources. Treat calls from domain services to external sub-functions as cleanly as calls from the external system environment to asset services. This will allow several kinds of "plug and play" flexibility when integrating asset base functionality into applications:

  — Depending upon the stability of the interface, asset base functionality could be tailored by integrating with different external components; and/or

  — Asset base functionality could be treated as a component that could be swapped, for example, with other implementations based on the same initial DOMAIN MODEL.

  However, in practice not all interfaces will be equally or simultaneously amenable to clean encapsulation. This will depend greatly on the relative maturity of the related domain. Graphical user interfaces (GUIs) have now evolved to the point where applications can be written with clean layers to the GUI; many other relatively "horizontal" functional areas have not evolved to this degree. In addition, there are interactions and constraints among the various interfaces.

7.2.2 Determine Internal Architecture Constraints

The ASSET BASE MODEL specifies the *overall* set of features that will be supported by the ASSET BASE, as well as a specific set of customer settings that will be targeted for utilizing the ASSETS. The mapping between them establishes which overall profile of features will be desired by each customer. In a single-system engineering context, the next step would be to allocate the required features to components of an overall system architecture. In the domain engineering context, there are several other dimensions of variability to be considered. In this task, engineers determine constraints on the *internal* structural variability of the asset base. This is complementary to the *external* constraints determined in the previous task.

The primary purpose of performing this task is to ensure that all constraints have been identified that could affect the predictability of different combinations of assets in independent use. A secondary benefit of the task is to identify opportunities for achieving greater levels of asset base utility for different customers by providing useful subsets of functionality that can be implemented and maintained in a cost-effective manner.

Engineer Asset Base

Scope asset base · Architect asset base · Implement asset base

Correlate features and customers · Prioritize features and customers · Select features and customers · Determine external architecture constraints · Determine internal architecture constraints · Define asset base architecture · Plan asset base implementation · Implement infrastructure · Implement assets

Exhibit 89. Determine Internal Architecture Constraints Process Tree

Approach

This task plays the pivotal role in *Engineer Asset Base* for determining the structure of the ASSET BASE. Although final decisions about partitioning and structuring the asset base are made in the *Determine Asset Base Architecture* task that follows, here is where the key information and insights about the *internal* asset base structure are established to support those decisions. This task produces information that helps in determining:

- How features will be clustered into coherent *feature ensembles:* sets of features that should be accessible *as a whole* (i.e., in the form of individual assets or well-defined groups of assets) by given customers in specific settings.

- How feature ensembles (and ultimately the assets that provide them) will relate to one another architecturally (e.g., component interactions/interfaces) and in feature content (e.g., specializations, decompositions).

- To what extent each feature ensemble will be *separately selectable:* elements of an architectural framework that can be included or deleted from system instances relatively independently.

A key technique used here is to take the feature sets in the ASSET BASE MODEL and factor them into subsets (in the extreme case, the set of all subsets, or power set) that become the basis for analyzing feature ensembles and their interrelationships. The subsets of domain functionality that are identified in this task can take two primary forms. Sets of sub-components that form restricted versions of the overall domain functionality are called *specializations* or *layers*. Other subsets of functionality are referred to as *components* or *subsystems* within the scope of the asset base. There may be complex interactions affecting the separate selectability of layers or components; a primary function of this task is to document these INTERNAL ARCHITECTURE CONSTRAINTS.

> *Example.* In the Outliner domain, an example of an internal architecture constraint might be a given customer's requirement that the outliner function in a read-write as well as a browse-only mode. This involves selection of more than individual feature variants because the implications pervade most of the functions in the application. In addition, some developers who would incorporate the outline module would need to fix this mode at development time; others would offer it as a start-up option to the end-user. These are internal constraints because they affect the topology, structure, interconnections and binding of aggregates of components developed in the asset base.

Workproducts

■ INTERNAL ARCHITECTURE CONSTRAINTS

This workproduct includes the following information:

* Identification of which feature ensembles are *desirable* in usage settings and which feature ensembles are *feasible* in development settings. Refines the information in the ASSET BASE MODEL.

* Key constraints pertaining to feature ensembles, such as whether they can be selected as stand-alone or separately selectable. The COMPONENT CONSTRAINTS TABLE worksheet template in Exhibit C-16 illustrates how this information can be expressed.

* Constraints and relationships between subsets/layered architecture variants and feature ensembles. The LAYERING CONSTRAINTS TABLE worksheet template in Exhibit C-17 illustrates how this information can be expressed.

When to Start

* The ASSET BASE MODEL has been specified in the *Scope Asset Base* sub-phase.

* Technology constraints that could reduce the set of structural variants considered have been identified.

Determination of the EXTERNAL ARCHITECTURE CONSTRAINTS need not be complete to begin this task.

Inputs

* DOMAIN INTERACTIONS. Source for component, sub-system, and specialization subdomains that determine key *internal* interface issues for the asset base.

* ASSET BASE MODEL. Source for the feature ensembles required by various customers for the asset base. These collectively represent the complete suite of functionality in the domain that must be available to given customers. The feature ensembles are allocated to specific domain settings (as for external constraints).

* ASSET BASE DOSSIER. Important if developers want to consider learning from legacy artifacts in identifying architecture constraints.

Controls

The following controls serve a role analogous to that which they play in *Determine External Architecture Constraints*.

* ASSETS OF INTEREST. Initial candidate partitionings of features into assets, based on the feature-customer analysis in *Scope Asset Base*. These may be useful in defining or validating feature ensembles.

* PROJECT CONSTRAINTS: Constraints on the domain engineering project that can impact the internal interfaces. These constraints can include:

    — Constraints on *project resources*, such as budget and schedule, as well as restrictions on

specific resources available for particular architectural options.

— Constraints on *available technology*, including restrictions on strategies for architecting the asset base (e.g., limitations in technology that affect options for supporting variability in architectures).

Activities

➤ Factor feature sets into feature ensembles

Map the *feature sets*, allocated to different customers and settings in the ASSET BASE MODEL, to *feature ensembles*. Each ensemble encompasses a set of features that needs to be accessed as a whole by practitioners within a given setting. Thus, decisions about which ensembles to consider are driven by anticipated customer needs.

The overall feature set associated with a given customer setting becomes its own feature ensemble by default. Other ensembles are identified by subsetting these overall ensembles, and also perhaps by combining features within two or more feature sets to form ensembles that are supersets of the original feature sets (this may be appropriate upon discovering economies of scope across customer settings). The result can be a different set than that identified in the ASSET BASE MODEL since that task did not systematically consider *all* subsets (or supersets) of selected feature sets. Note, however, that only those sets identified as part of the DOMAIN MODEL are considered. (If new subsets or supersets are discovered or proposed, they ideally should be flowed back through the *Resolve Domain Model* task in the *Model Domain* phase.)

> *Example*. For a development setting, there may be design features to provide code components in either functional or subroutine form. If a typical user would call only one form of component, these two features could be partitioned separately. If the same engineer would need access to both forms of component, these two features become effectively part of one feature ensemble.

Each ensemble selected for support will be implemented by some mix of assets using component or generative techniques. Record the results as part of the INTERNAL ARCHITECTURE CONSTRAINTS.

➤ Determine component subsystems

Given a feature ensemble F, each of the ensembles F' which are a subset of F but not a subset of any other F' are considered to be **component subsystems** of F. Document the constraints on ensembles of this type. The COMPONENT CONSTRAINTS TABLE worksheet template in Exhibit C-16 suggests a format for capturing this information.

Each of these component subsystems should be evaluated for separate selectability. Separate selectability refers to a means of achieving variability in system instances by encapsulating subcomponents of an architectural framework so that they can be included or deleted from system instances relatively independently. For any system S with subsystem S1, consider the following variant instantiations:

- The sub-component as a stand-alone variant (S1);

- The system with the sub-component *masked* (S without S1);

- The system with the sub-component (S, including S1).

For each component subsystem, consider the following architectural issues:

- Can the component be masked? That is, can a system variant be produced without the component's functionality included?

- Can the interface to the component be encapsulated to allow for alternative implementations of the component?

    Example. In the Outliner domain, could a sub-system to generate heading level numbering be masked out of a system version where this function would not be required? Could the program be parameterized in such a way that user-written heading numbering routines could be provided?

► Determine specialization/layered subsystems

Given a feature ensemble F, each of the ensembles F' which are a subset of F and (optionally) a subset of another F' are considered to be *specialization* or *layered subsystems* of F. These can be viewed as implementing the functionality in F at varying degrees or levels (i.e., via progressive restrictions of functionality). Document the constraints on each ensemble of this type. The LAYERING CONSTRAINTS TABLE worksheet template in Exhibit C-17 suggests a format for capturing this information.

It may be helpful to document the semantic relations between specializations using a *layered architecture* approach for the feature sets contained within the scope of the ASSET BASE MODEL. In a layered approach, an inner kernel with minimum or core functionality is surrounded by successive layers, each of which provides enhanced functionality in some respect. For each candidate architectural layer, consider the following issues:

- Will the layer be separately selectable as an instantiation of domain functionality? (See the discussion on separate selectability in the Guidelines section below.)

- If instantiated as a separately selectable variant, will the layer need to be *optimized* with respect to its own internal layers?

- If incorporated into outer layers, will there be separate interfaces to the layer? For example, many interactive programs also provide application program interfaces (APIs). An asset base could be architected so that there are several potential API levels available; not all will be desired in every instantiation.

► Consider internal coupling trade-offs

Consider trade-offs of strong versus loose coupling of assets within the structure of the asset base. Strong coupling will increase the internal reuse and modularity of assets, but will create more asset-to-asset interdependencies. Loose coupling will preserve greater autonomy of separate assets, but usually with additional overhead. The burden may fall as much on the configuration management and version control mechanisms available as on the implementations themselves.

While cohesion versus coupling trade-offs are familiar from general software engineering, they present special problems here because each asset may be required to perform within a number of different application instances. These requirements will impose significant constraints on the ASSET BASE ARCHITECTURE.

Some of these constraints and trade-offs can be derived from semantic information about the features mapped to each ensemble. Since the asset base architectural information defined at this

stage links only the feature ensemble specifications as a whole, it can record only the general fact of this dependency between the two feature ensembles.

> *Example.* Suppose one feature ensemble represents a family of variant algorithms used within the domain of focus, while another ensemble represents variants of a key data structure used in the algorithm calculation. For any instantiation of the algorithm assets there may be restrictions on what variants of the data structure asset(s) can be instantiated as part of the same system.

➤ Validate internal constraints

- Validate that each ensemble has been allocated only once (see Guidelines) as a component or layer.

- Validate the trade-off analysis for internal consistency.

- Validate assumptions of utility/feasibility for individual ensembles with asset base customers; also compare to the ASSETS OF INTEREST derived from utility/feasibility analyses.

When to Stop

- The feature ensembles defined in the ASSET BASE MODEL have been allocated to internal architectural categories.

- The trade-offs for each ensemble in terms of masking, separate selectability, and optimization of internal structure have been documented.

The final selection of encapsulation, optimization, and separate selectability attributes to support are determined in the subsequent task, *Define Asset Base Architecture.*

Guidelines

- Select a small set of combinations for analysis. Separate selectability makes the architectural problem inherently more complex for reuse than in the design of a single system. The *same* component may appear as an element of several overlapping, separately selectable subsystems obtainable from the overall architecture. In the most complex case, this could lead to a combinatorial explosion of architectural variants for a given component topology.

  > *Example.* Few compilers are architected so that the lexical analyzer, parser, and/or code optimizer are all easily accessible as separate utilities. In engineering a single system, each component can be designed to be modular, perhaps even separately selectable on an individual basis. (This is rarely applied systematically; for example,).

  Often in domain engineering no one way of partitioning the system into components will meet the needs of multiple potential application contexts. Modelers need to select a small subset of these combinations to support as separately selectable for any practical asset base. If desired, consult the DOMAIN INTERACTIONS, ASSET BASE DOSSIER, and/or ASSETS OF INTEREST to filter out subsystems that are not likely to be of interest.

- Components and specializations are dual perspectives. The distinction between components and specializations of the asset base functionality can become blurry in cases where a large proportion of functions are provided. That is, some ensembles can be difficult to classify clearly in one category or another. A system with one component masked out may be classified as a specialization or restricted-functionality variant of the system.

Do not spend much effort in looking for the "right" allocation. As long as each feature ensemble has been covered in the scope of at least one activity, the allocation is not critical to the main purposes of the task.

7.2.3 Define Asset Base Architecture

The two previous tasks in this sub-phase were to *Determine External* and *Internal Architecture Constraints* for the ASSET BASE to be developed. The primary purpose of the task, and the rationale for performing it separately from and subsequent to the earlier tasks, is to enable decisions to be made by integrating both internal and external constraints simultaneously and in a managed way. The benefit is that higher-quality architectural strategies will be revealed that may not have been chosen via more informal methods. In particular, this strategy is oriented towards managing the potential complexity of highly variable architectures, which makes the process robust enough for a wide variety of domains.
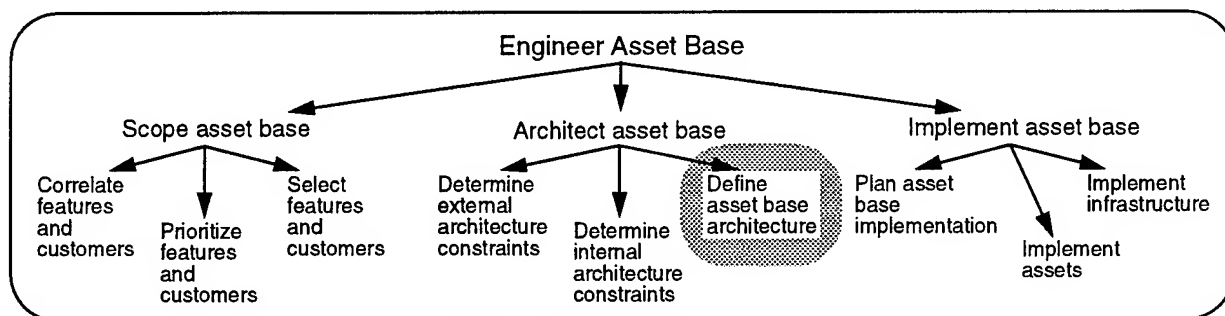
Exhibit 90. Define Asset Base Architecture Process Tree

Approach

In this task the final selection is made for a range of external and internal architectural choices:

- the range of external interfaces to be supported;

- the degree of coupling and other interconnection strategies to be supported for each interface;

- which components within the scope of the asset base will be implemented to be separately selectable, masked, optimized, etc.

These results are documented in the ASSET BASE ARCHITECTURE.

Workproducts

■ ASSET BASE ARCHITECTURE

The ASSET BASE ARCHITECTURE models the range of variability among system architectures or configurations that can be derived from feature combinations in the ASSET BASE. It includes the rules, relationships, and constraints among ASSETS that govern how they can be combined and instantiated to produce application artifacts supporting specific feature ensembles. The architecture supports separately selectable feature ensembles. It should be as implementation-neutral as possible.

■ Asset Specifications

Asset Specifications contain specifications for Assets that implement the selected feature ensembles. The Asset Specifications include the required range of variability in the Assets, and constraints and interdependencies with other Assets. Each of these specifications is subject to an independent technology selection process in the next sub-phase, *Implement Asset Base.*

When to Start

- The Asset Base Model should be complete. The trade-off analysis performed in this task attempts to optimize for the full set of features and customers to be supported. The task activities would need to be repeated if these change significantly.

- Initial results of the other two tasks in this sub-phase should be available. At a minimum, enough constraints should have been documented to highlight the trade-off issues involved.

Inputs

- External Architecture Constraints. Key input to the architecture trade-off analysis and selection of variants to be supported.

- Internal Architecture Constraints. Key input to the architecture trade-off analysis and selection of variants to be supported.

- Asset Base Model. Provides input for prioritizing the variants to be selected.

Controls

- Assets of Interest. Initial candidate partitionings of features into assets, based on the feature-customer analysis in *Scope Asset Base*. These may be useful in guiding final decisions for allocating features to assets.

- Project Constraints: Constraints on the domain engineering project that can impact architectural decisions. These constraints can include:

  — Constraints on *project resources*, such as budget and schedule, as well as restrictions on specific resources available for particular architectural options.

  — Constraints on *available technology*, including restrictions on strategies for architecting the asset base (e.g., limitations in technology that affect options for supporting variability in architectures).

  Note that project resources should *not* be considered a control on the Asset Specifications. Development of asset specifications does not constitute a commitment to develop all assets specified within the scope of the domain engineering project.

Activities

System Modeling supporting methods (particularly those pertaining to software architecture) play an important role throughout the following activities.

330

> ➤ Map internal to external architecture constraints

Determine consistencies and potential inconsistencies between internal and external constraints. Resolve the inconsistencies, perhaps by iterating among the other tasks in this sub-phase (or, where appropriate, treat them as allowable variations). Locate entry and exit points to and from the internal architecture for each external interface.

> ➤ Consider alternative architecture approaches

Consider alternative architectural approaches to accommodate the specific feature and (system) architectural variants committed to for the ASSET BASE MODEL.

Bearing in mind that the ASSET BASE ARCHITECTURE is the focus of this design activity, there are several architectural approaches that are worth particular attention as strategies for accommodating variability. Each can be seen as a response to specific challenges of variable architectures.

- The ASSET BASE ARCHITECTURE may be designed to produce application architectures that are layered, component-oriented, or both (or perhaps any of these, depending on the variability of the requirements and constraints within the target customer settings).

- The architecture may include coarse-grained assets (potentially restricting variability, but easing system composition/generation) or fine-grained assets (supporting numerous and perhaps subtle variations, but complicating composition/generation), or both (including the possibility of a coarse-grained generator that composes fine-grained component assets).

- A component or subsystem asset within the asset base may be configured to support multiple feature ensembles.

> ➤ Consider variability requirements for system architecture partitions

Assume that the features in the ASSET BASE MODEL have been allocated to initial feature ensembles and that these ensembles have been partitioned into component and layered subsystems. These feature ensemble partitions reflect varied customer requirements and will likely impose variability requirements on the partitioning of functionality within individual system architectures to be supported by the asset base. Document these requirements.

> ➤ Develop unified asset base architecture

After performing trade-off analysis on the architectural alternatives considered, select and document the best alternative. The resulting ASSET BASE ARCHITECTURE structures the set of feature ensembles that make up the functional scope for the asset base.

> ➤ Allocate features to architectural components

This can be done in several steps. Since features were previously mapped to partitions, first map these partitions to architectural components, then derive the feature-to-architecture mapping.

> ➤ Transform allocated features to asset specifications

Interpret the allocated features in terms of interfaces to (and from) asset functionality or services, constraints on asset behavior, constraints on asset development, and other key factors. Describe these in a specification language of choice.

➤ Develop utilization profile for each asset specification

Characterize the settings in which each asset is targeted for use and the architectural niche into which the asset fits (this can be viewed as the shape of the hole into which the asset "peg" must fit).

➤ Validate asset base architecture and asset specifications

- Formal validation: It is difficult to test the ASSET BASE ARCHITECTURE independently of the ASSETS it supports. Guidelines for formally evaluating architectures are still emerging. Criteria might include: completeness of the architecture with respect to the targeted solution space, as represented by the domain exemplar systems the architecture is committed to cover; consistency of the architecture models (both with themselves, and in relation to other models of the domain); and parsimony (e.g., do models allow specification of assets not intended for support?)

- Customer review: Allow potential customers to review the final ASSET BASE ARCHITECTURE, particularly from the standpoint of the ASSET SPECIFICATIONS and their mapping to feature ensembles. It may be helpful to have a prototype of a potential utilizer's interface to the asset base (i.e., some portion of the ASSET BASE INFRASTRUCTURE) as a way of getting this feedback. Customers would be, in effect, browsing the specifications only, so they won't be able to validate the assets themselves.

When to Stop

- The ASSET BASE ARCHITECTURE and a set of ASSET SPECIFICATIONS have been defined that are implementable and address the needs identified within the targeted customer settings.

Guidelines

- Defer implementation choices where possible. Allow for different assets to be implemented with different technology choices (e.g., generative, constructive). This will support heterogenous domains where one across-the-board technology approach is not suitable. Local application of technology can support feasibility in some cases. e.g., smaller generative tasks are more tractable.

  This strategy will also support evolution over time. Individual components may evolve toward generative implementations with repeated usage and refinement. The architecture should support this evolution as seamlessly as possible. Strive for independence with respect to evolving standards and conventions.

7.3 Implement Asset Base

The previous sub-phases in *Engineer Asset Base* have produced an ASSET BASE MODEL, ASSET BASE ARCHITECTURE, and ASSET SPECIFICATIONS; these workproducts, taken together, constitute the equivalent of requirements, architecture, and high-level design for the assets in the asset base. The primary purpose of the *Implement Asset Base sub-phase* is to implement that portion of the specified asset base that is essential in order to transition it into use.
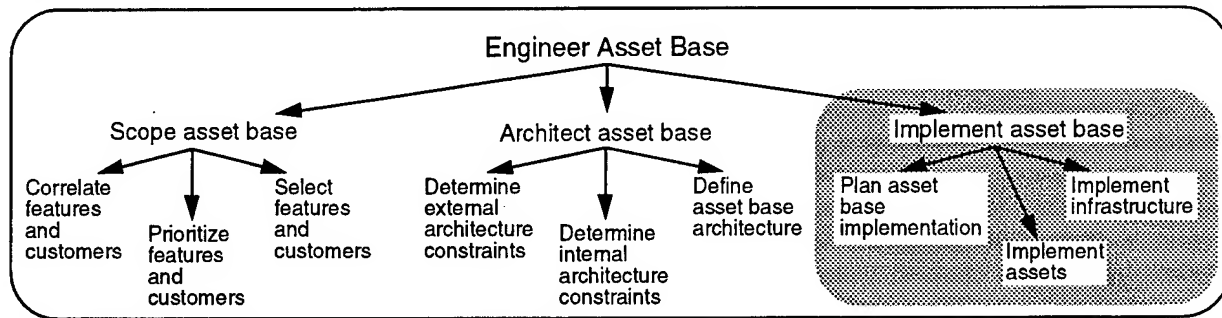
Exhibit 91. Implement Asset Base Process Tree

Approach

This sub-phase is strongly dependent on supporting methods. The activities in this sub-phase are closely related to software engineering practices that are likely already in place in a software development organization. The tasks in this sub-phase (particularly *Implement Assets* and *Implement Infrastructure*) offer ODM-specific guidelines for applying the implementation methods, processes, and tools selected (and often already in use) by the organization.

Results

The primary results of the *Implement Asset Base* sub-phase are:

- A set of ASSETS that are the directly reusable entities within the ASSET BASE. In the system and software engineering context, an asset is a discrete system/software entity that encompasses some set of features (typically, feature variants) and can be instantiated to supply those features (or a subset) to one or more application artifacts.

- An ASSET BASE INFRASTRUCTURE that supports application engineers in using the ASSET BASE to develop and maintain their applications.

History has shown that even the best assets may sit unused if there are no crisp, clear mechanisms to support their use. Thus, development of the ASSETS (and of the ASSET BASE as a whole) is coupled with development of an ASSET BASE INFRASTRUCTURE to assist application engineers in applying the domain assets to their application problems.

Process

As shown in Exhibit 92, there are three major tasks in *Implement Asset Base*:

- The *Plan Asset Base Implementation* task involves selecting the technology to apply for each asset, planning implementation phases, and planning the test and validation strategy for the assets.

- The *Implement Assets* task carries out the core work of implementing the assets, in accord with the ASSET SPECIFICATIONS, and supported by the ASSET BASE INFRASTRUCTURE.

- The *Implement Asset Base Infrastructure* task implements support mechanisms for asset base customers, integrates the assets with general infrastructure mechanisms, and addresses any other asset base-wide implementation tasks.

Guidelines

- <u>Plan sufficiently before implementing</u>. In the most basic sequence the *Plan Asset Base Implementation* task is completed before the other two tasks commence. It is helpful to plan the implementation *phasing* strategy in particular before extensive implementation of particular assets, because the phasing strategy may allow for deferring implementation of certain assets, or ways to leverage certain other assets by developing them early.

- <u>Assets may be implemented individually</u>. In some other respects, though, this sub-phase deals with processes that can occur in the development of each individual asset; e.g., technology selection choices are made somewhat independently for each asset. So the entire *Plan Asset Base Implementation* task does not need to be complete before the other tasks begin.

- <u>Consider early prototyping of assets</u>. Taking the last point one step further, there may be advantages to implementing at least a few assets early in the sub-phase to validate implementation approaches being followed in the plan as a whole.

- <u>Localize the impact of shifts or evolution in implementation technology</u>. For example, conversion of an individual family of static components to a generator should not effect the technology associated with other components. This facilitates later migration to alternative

Exhibit 92. Implement Asset Base IDEF₀ Diagram

334
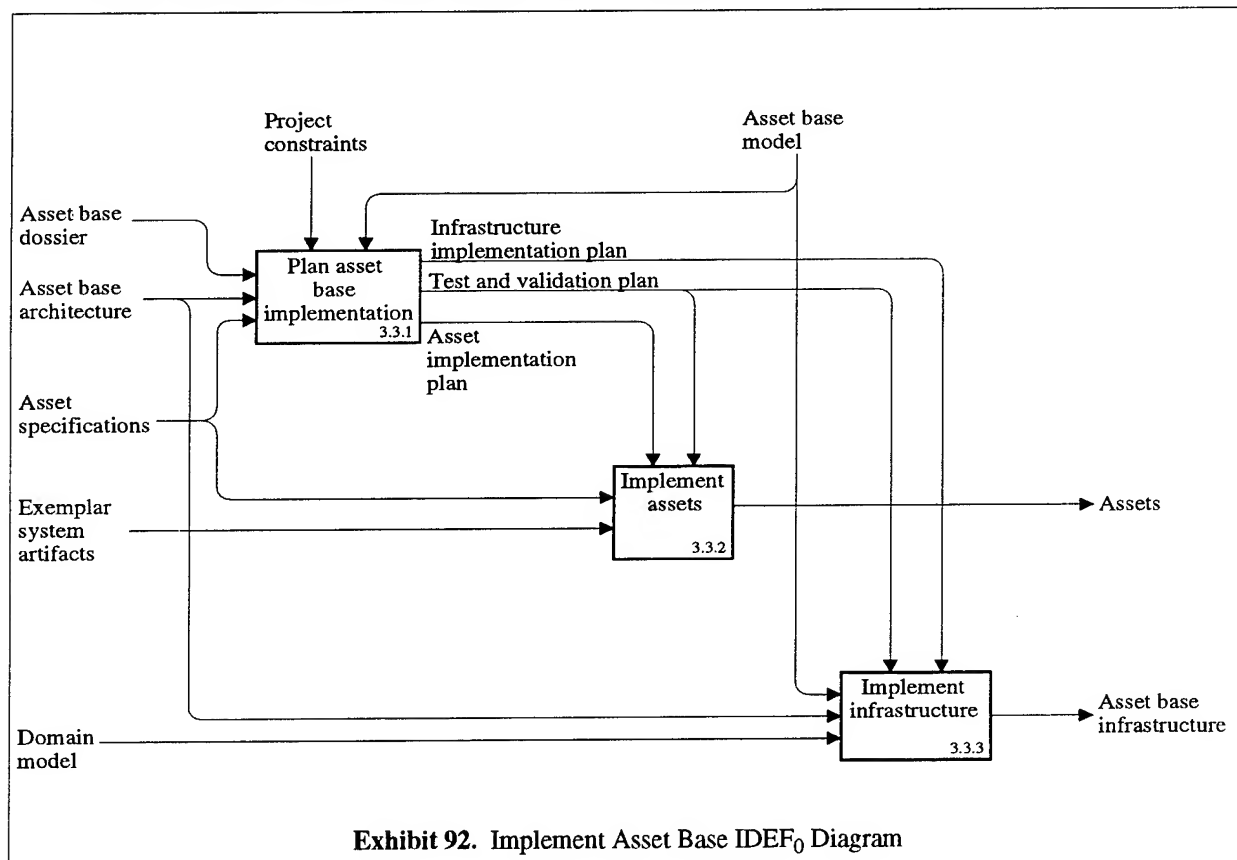
implementation strategies with minimal disruption to overall asset base structure.

7.3.1 Plan Asset Base Implementation

The primary purpose of *Plan Asset Base Implementation* is to select implementation strategies for individual assets in the context of the ASSET BASE ARCHITECTURE as a whole. Technology choices can be made throughout the ODM life cycle; for the most part, these are considered part of general infrastructure planning and tailoring activities, and hence not included in the core process model. However, late binding of asset implementation technology choices is a significant feature of the ODM method. Deferring technology selection into the *Engineer Asset Base* phase helps ensure that technology choices fit domain characteristics, not merely implementors' past experience and technical biases. By deferring technology choices for individual assets, to the degree possible, until after specification of the ASSET BASE ARCHITECTURE, finer granularity in technology choices can be obtained (in contrast to, for example, an architecture-level decision to use an across-the-board generative or component-library approach).
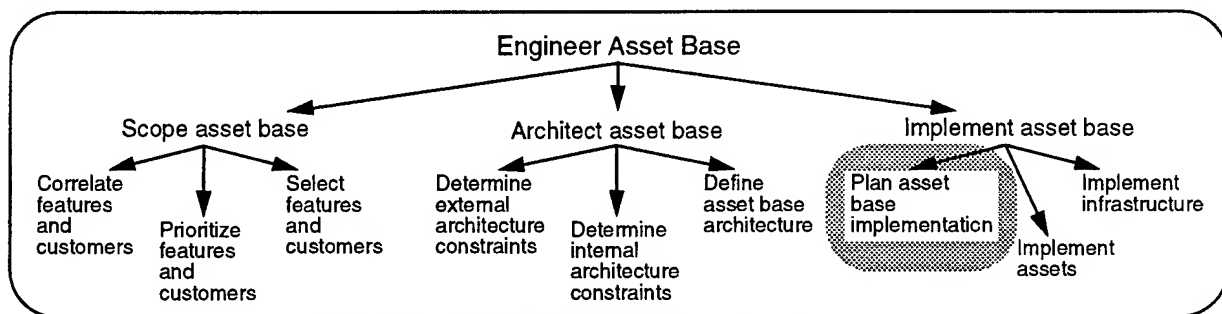
Exhibit 93. Plan Asset Base Implementation Process Tree

Approach

To whatever degree technology-independence in the ASSET BASE ARCHITECTURE has been achieved, planning asset implementation as a unified up-front task facilitates coordinated rather than piecemeal use of heterogenous technologies, by considering interactions of technology choices within the ASSET BASE as a whole. In addition, it allows for systematic consideration of alternative (e.g. constructive versus generative) approaches on a *per-asset basis*.

Other major factors to consider in this task include the following:

- Maximize use of legacy artifacts. Assets may be developed using a combination of newly developed workproducts and adaptations of existing artifacts characterized during the MODEL DOMAIN phase.

- Obtain economies in implementation. Determine the optimum strategy for incrementally implementing the assets and separately selectable subsystems; e.g., by preserving interim versions under separate configuration management so that the version is always obtainable from the asset base and reuse infrastructure.

- Factor application engineering project constraints. Select the best strategy for transitioning results of reuse engineering into application engineering groups at varying points in their project life cycles.

Workproducts

■ ASSET IMPLEMENTATION PLAN

Details of the technology to be used and the implementation schedule for each ASSET. Includes information such as:

- A description of the technology to be used and a general plan for how the technology is to be applied to implement each asset.

- A schedule for asset implementation that is correlated with estimated schedules for customer applications.

- A phased asset implementation and evolution strategy that describes which assets will be implemented when and the phases they may go through, based on customer need, technology availability, feedback cycles, and so on.

■ INFRASTRUCTURE IMPLEMENTATION PLAN

A plan for implementing the ASSET BASE INFRASTRUCTURE. The INFRASTRUCTURE IMPLEMEN-TATION PLAN includes technology selections and scheduling constraints for infrastructure development relative to development of ASSET releases. The infrastructure may be specifically developed for the ASSET BASE or may be based on available technology that is applied to the ASSET BASE. In general, individual ASSET IMPLEMENTATION PLANS will be affected by the choice of ASSET BASE INFRASTRUCTURE.

■ TEST AND VALIDATION PLAN

A plan for how individual ASSETS, the ASSET BASE INFRASTRUCTURE, and the ASSET BASE as a whole will be tested and validated. This plan is based on the types of assets to be implemented, the implementation techniques selected, and the phased development plan. It may involve planning for the development and use of asset test cases (e.g., to test if an asset supports the desired range of variability) and of *test* (and *test plan*) *assets* that can be instantiated to support testing of the system artifacts derived from other assets (note that the detailed implementation of such assets should be covered in the ASSET IMPLEMENTATION PLAN).

When to Start

- Planning for asset base implementation should not commence until the individual ASSET SPECIFICATIONS and the overall ASSET BASE ARCHITECTURE have been developed and validated.

  The primary advantage of the planning step involves making technology decisions in a coordinated way for the entire asset base, rather than on an asset-by-asset basis.

- Black-box testing strategy and even test plans could be derived for variants selected as early as completion of the ASSET BASE ARCHITECTURE and even, theoretically, the ASSET BASE MODEL.

  This approach would be similar to creating test plans directly from requirements or high-level designs in non-reuse-based software development. Such test plans might be fairly abstract, as they would need to be formulated independent of assumptions regarding the ASSET BASE ARCHITECTURE or implementation technology; e.g., a test plan implementation would be quite different if the functionality to be tested were implemented as a family of components

336

versus a generator.

Inputs

- ASSET SPECIFICATIONS. Feature profiles of individual assets to be implemented. A strategy is developed for each specified asset.

- ASSET BASE ARCHITECTURE. Overall architecture of the asset base. A strategy for developing the ASSET BASE INFRASTRUCTURE is derived from this workproduct.

- ASSET BASE DOSSIER. Contains utility and feasibility data, as well as traceability to possible prototype artifacts for reengineering.

Controls

- ASSET BASE MODEL. Used to bound the features and customers and interface constraints considered.

- PROJECT CONSTRAINTS: Constraints on the domain engineering project that can impact asset base implementation. These constraints can include:

  — Constraints on *project resources*, such as budget and schedule, as well as restrictions on specific resources available for particular implementation options.

  — Constraints on *available technology*, including restrictions on strategies for implementing the asset base (e.g., limitations in technology that affect options for supporting variability, such as availability of or expertise in generative techniques).

Activities

➤ Identify customer constraints on technology selection

Potential customer settings must be characterized more fully with respect to the actual implementation forms and packaging of the reusable assets to be provided. Some knowledge of the current forms of system workproducts (studied as artifacts) has been captured during the *Describe Domain* sub-phase. This information (as now contained in the ASSET BASE MODEL and ASSET BASE DOSSIER) can be reexamined and refined to express constraints on the implemented form for assets.

➤ Consider alternative asset implementation strategies

For each ASSET SPECIFICATION do a trade-off analysis, particularly on the benefits and risks of a component-oriented versus generative implementation approach. This trade-off analysis must consider several factors, including:

- The mechanisms for supporting variability in the implementation language (e.g., an object-oriented versus functional language);

- The level of sophistication in configuration management that will be required in the ASSET BASE INFRASTRUCTURE;

- The potential for combinatorial explosion in feature combinations desired for the asset.

➤ Select asset implementation technologies

One goal of previous sub-phases in *Engineer Asset Base* is to produce ASSET SPECIFICATIONS that reduce the *Implement Assets* task to a tractable problem for software engineering design techniques. Once this has been accomplished, the implementation technology applied is considered part of the supporting methods documented in Section 8.0. In general, the software engineering methodology used within the organization can be applied, with the following provisos:

- Adapt methodologies to reuse existing artifacts where possible.

- Adopt or develop standards for use across similar implementation tasks. For example, the more separate generative tasks are required, the more important it is to have a standard meta-generation technology.

➤ Update ASSET BASE ARCHITECTURE and ASSET SPECIFICATIONS

Incorporate technology decisions into updated versions of both these sets of workproducts. The result can be considered *technology-independent* and *technology-dependent* versions respectively. Maintain these updated versions separately and in parallel with the original versions, if possible. This will facilitate possible roll-back later if technology decisions need to be considered, and will provide a "technology-hiding" specification layer that can be used in external reference specifications, so that flexibility is retained to evolve technology choices later.

➤ Consider customer constraints on schedule

Consider the current life cycle phase for application projects that are potential utilizers of the assets to be developed. The overall plan should reflect opportunities and conflicts stemming from the schedule of each utilizer project. Adjust the schedule or phasing of implementation if necessary to coincide with these phases. This can make the difference between assets being used or being passed by, whatever their technical merits.

Place the anticipated project schedule with milestones for completed assets on a common time line with the anticipated schedules of potential customer applications. A separate phasing strategy may be required for each existing or anticipated system into which domain assets are to be incorporated.

> *Example.* One project might be in an early requirements definition and negotiation phase, where initial versions of domain assets could be used in a prototyping capacity. Another project may be almost through with detailed design, and therefore only be able to make use of thoroughly tested components that offer a close match to the functional profiles assumed by the current architecture. Each life cycle entry point presents different challenges.

➤ Plan development stages

Based on the schedules developed above and the feasibility estimates for the various ASSET SPECIFICATIONS, develop the detailed staging strategy for implementing assets. Consider factors such as requirements/opportunities for migrating or back-filling assets into existing systems, use of existing systems as testbeds/validation suites, and advantages of early release of certain sets of assets to facilitate adoption with particular customers.

This staged approach can support an incremental strategy for developing ASSETS that "flesh out" different portions of the ASSET BASE ARCHITECTURE over time. Note, however, that decisions about which ASSETS to implement when should be made strategically, based on market needs,

rather than for reasons such as developer convenience. This implies that incremental development of the ASSET BASE should be done in strategic "chunks" involving coherent sets of assets that address specific market needs.

On a per-asset basis, the staging strategies can include any of the following options:

- *Prototype assets.* Assets implemented early enough to serve as validation of the DOMAIN MODEL, in particular the adequacy of the feature language as a specification of required functionality. Assets can also be done as prototypes in order to assess the technical feasibility of particular asset implementation techniques, methods, and tools.

  None of these types of prototype assets are expected to be incorporated into the final asset base. The assumption is that final assets must be implemented in the context of an overall asset base architecture.

- *Assets to validate architecture.* A set of assets may be implemented that, taken as a whole, validate some aspect of the ASSET BASE ARCHITECTURE. For example, a set of routines may implement a given algorithm across a range of optimizations for diverse engineering factors. By implementing the entire set of these assets, the soundness of the technical approaches for supporting and demonstrating different optimization priorities can be validated. This still need not constitute implementation of the entire asset base.

- *Assets to validate infrastructure.* Similarly, a set of assets may be selected that, taken together, will allow for validation of key aspects of the ASSET BASE INFRASTRUCTURE (to the extent that these aspects are distinct from purely architectural factors).

- *Assets to validate utilization.* A set of assets adequate for incorporation into a complete system is probably highly desirable before attempting a full-scale implementation of all assets in the asset base. The key is that enough of the assets need to be implemented that system developers can experience the direct impact of having the assets available for use.

- *Assets implemented by asset utilizers.* Some assets may be included on an "implement as needed" basis. That is, the ASSET SPECIFICATIONS (and possibly even the technical design, such as choice of programming language and encapsulation techniques) are prepared during domain engineering; but the actual implementation of the assets is left to the system developer when they are needed.

The ASSET IMPLEMENTATION PLAN should take into account decisions on implementation technology. In component-based engineering, the plan should specify the sequence of various builds or versions of domain assets to be developed. In generator-based engineering, the plan might detail plans for successively introducing greater levels of semantic completeness in the generator languages.

➤ Plan for asset evolution

In addition to applying a staged approach to initial asset implementation, it can be useful to plan at least a general approach for evolving assets in response to changing marketplaces, customer needs, technology availability, user feedback, and so on. Some of the forms of evolution that it may be useful to account for in the plan include:

- Making incremental improvements to assets in response to application engineers' feedback from tailoring assets and integrating them into applications.

- Shifting asset implementations to a more sophisticated, mature, or easier-to-use technology base.

- Aggregating assets into new, larger-scale assets that reflect frequently observed reuse patterns within segments of the customer base.

➤ Plan asset base testing and validation

Based on the types of assets to be implemented, the implementation techniques selected and the phased development plan, plan the strategy for testing and validation of both individual assets and the asset base as a whole. Document this in the TEST AND VALIDATION PLAN. The following paragraphs highlight some points to consider.

Test assets for conventional use

Assets must be tested in a manner consistent with their conventional use within a domain application. There are stronger incentives to apply thorough testing techniques to assets that will be reused multiple times, in multiple applications. A reusable code component must be tested in analogous way, but more rigorously, than a component written for a single point of use; e.g., the component may need to be tested against several parallel sets of requirements, rather than one set of requirements.

The component may need to be tested more exhaustively, to satisfy a higher level of trust. It may need to be boundary-tested more thoroughly, so that its behavior when used in unanticipated settings can be adequately predicted. Note that this is not equivalent to saying components must be maximally robust. Components can be engineered to make as strong or as weak assumptions about usage settings as is appropriate in the domain. The job of testing in asset base engineering is to verify that the component performs exactly as specified; i.e., testing for undocumented excess functionality as well as adequacy. It may also require system benchmarking tests at lower levels than would normally be required during system test, so that a library of variant components can be adequately characterized with regard to performance trade-offs.

Additional requirements for asset testing

Conventional software testing practices are oriented towards implementations (i.e., code). Asset bases can contain assets from across the software life cycle. The TEST AND VALIDATION PLAN should address how the various categories of reusable assets will be tested. Testing methods must be adapted to non-code artifacts that are to engineered for reuse to validate that they adequately support the intended range of variability. For example, if design schemas are to be included in the asset base, some means of validating the design schemas with regard to their intended scope of applicability must be specified. The design component validation must ensure the design complies with relevant system constraints on form and structure.

In addition, assets may be developed using a spectrum of reuse engineering techniques, including component design, generative reuse, table-driven support for variability, macro expansion or hand-tailored templates. Each technique may require different techniques for testing and validation.

Use model semantics in testing

Reusable assets must be tested, or qualified, with respect to their placement or classification within the ASSET BASE MODEL. A component that functions appropriately in a single usage setting may prove unacceptable when classified within the domain model because implicit contextual assumptions made in its implementation may violate conditions assumed by the model. Assets can fail by providing too much functionality, if their position in the asset base is intended to provide a specific set of features relative to other assets.

The TEST AND VALIDATION PLAN can take advantage of the defined semantic relationships between asset variants. Inheritance-based network modeling techniques allow relatively strong assumptions about the behavior of an asset based on its position within the network. Each category within the model can be associated with a set of possible asset implementations satisfying a particular set of requirements or conditions. Specializations within the network must satisfy the same set of requirements, plus possible additional constraints. Thus much of the testing protocol can be reused in multiple places within the model.

This will apply to overall system versions as well as specifications for individual assets. Because different versions or system configurations are characterized explicitly in terms of addition of features or relaxation or tightening of semantic restrictions, the TEST AND VALIDATION PLAN for the asset base can evaluate each more specialized version's compliance with new requirements and maintained compliance with old requirements.

It may be desirable to engineer some asset test plans as "test plan assets". Such assets must be designed for a specified range of variability to be incorporated within the testing phase of different applications. (Of course, theoretically, such "testing assets", like all other assets, should be tested before inclusion in the asset base. This means that in principle test assets will require their own test plan!)

➤ Validate plans

- Validate individual asset implementation plans for consistency relative to the features in individual ASSET SPECIFICATIONS.

- Validate technology selection choices for over-all coordination and consistency. For example, standardized choices should be made for similar implementation approaches within the asset base. Not all assets need to be built with a generator; but three different generative tools should not be specified without good rationale for the diversity.

- Prototype to validate technology selections. A good rule of thumb would be to prototype at least one asset using each candidate technology considered.

- Use asset base during development. Asset developers should become utilizers of the asset base wherever possible (except where this violates constraints on component sharing imposed for architectural reasons, as noted above).

The final validation of the ASSETS, ASSET BASE ARCHITECTURE, and ASSET BASE INFRASTRUCTURE as a whole comes through utilizing them to build an application. This can be considered a kind of acceptance testing.

When to Stop

- Implementation technology selections have been made for all assets to be developed as part of this *Engineer Asset Base* phase.

- A schedule for implementing assets has been developed that is doable given PROJECT CONSTRAINTS and the estimated feasibility for the assets involved.

Guidelines

- Consider component sharing trade-offs. A primary design issue to be considered in this phase is the potential sharing of components among assets. Greater degrees of sharing will lead to a more strongly coupled asset base, which may minimize development effort and consolidate

the workproducts produced. However, such strong coupling may work against the goal of separate selectability of subsystems and subsets of assets, as defined by the architectural variants to be supported.

- <u>Consider inside-out versus outside-in strategies</u>. In domain engineering, a set of components that differ by progressively adding functionality (in terms of the features of the domain model) can be implemented via several alternative strategies. An incremental, or "inside-out" approach implements an initial system version with minimum functionality, then incrementally extends the functions with subsequent versions. maps each broader set of features to a later version, with new functions added sequentially.

  However, it can sometimes be more advantageous to take an "outside-in" approach by implementing the "richer" system version first, then creating subsequent versions that suppress, mask or strip away excess functionality to arrive at the "leaner" system variants required by the ASSET BASE ARCHITECTURE. One challenge in this approach is to make the functional reductions in such a way that the variants can be maintained as parallel versions. These complementary inside-out versus outside-in alternatives are a recurrent trade-off issue throughout the domain engineering life cycle. In typical single-system development, there is a much smaller "look-ahead window" for optimizing development in this way; hence the design trade-offs are less often confronted directly.

- <u>Consider bootstrapping strategies</u> for use of early components later in the development cycle. The ASSET IMPLEMENTATION PLAN should capitalize on opportunities to build certain capabilities early in the implementation process that can be used later as well as becoming part of the final asset base. Besides achieving economies in development, this helps to validate tools and assets through direct and immediate usage.

- <u>Consider scaffolding layers</u>. A layered architecture may have been adopted for the overall asset base; e.g., a set of base components with layers of composite functions available above this level. In implementation, it is sometimes desirable to specify additional, "thinner" layers within the major architectural layers. Some additional layers may serve as scaffolding or throwaway layers, interim representations upon which desired persistent architectural layers are constructed. Other layers may be used as "envelopes" to facilitate incremental phasing of new assets and subsystems into existing systems.

- <u>Consider test plans as assets</u>. White-box test plans are dependent on implementation technology choices and phasing incorporated into the ASSET IMPLEMENTATION PLAN. Test plans for phased releases specified in the ASSET IMPLEMENTATION PLAN that represent variants to be maintained under permanent configuration management might usefully be engineered as "test assets" in their own right. Test plans, test cases and test results can be stored explicitly in the asset base, where they can provide auxiliary examples and help to document the intended behavior of the assets. These test assets can be integrated into asset qualification functions eventually used by asset base managers or asset utilizers in retrieving and evaluating existing assets and adding new assets to the asset base.

7.3.2 Implement Assets

The earlier steps of the ODM life cycle are intended to result in specifications that constrain the potential variability in implementing reusable assets down to a tractable engineering problem. Software development methods and tools with which the organization is familiar and comfortable can now be applied to the task of implementing the assets themselves.

The primary result of this task is the ASSETS themselves. The long journey through the domain engineering life cycle that leads to these products has enabled them to be engineered with consid-
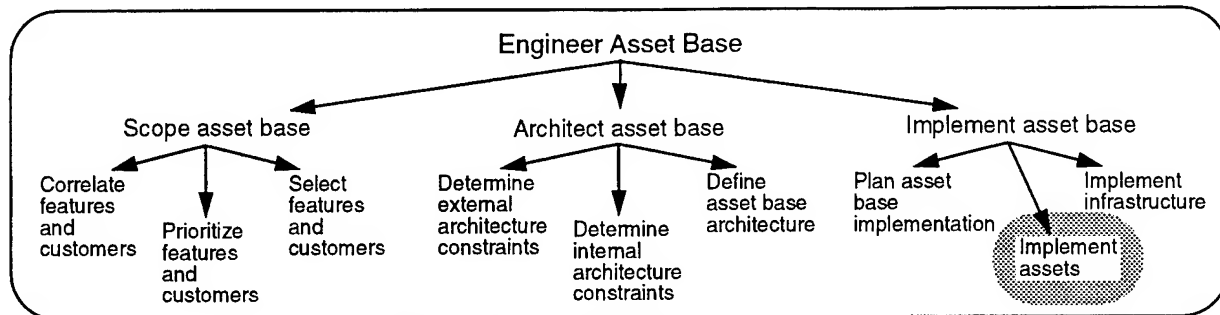
Exhibit 94. Implement Assets Process Tree

erable advantages over products of less disciplined approaches. The specification facilitates unstructured browsing and *ad hoc* queries by asset base customers, since it is mapped to the terminology of practitioners in the domain. It is also amenable to automated support mechanisms for retrieval, configuration, and interchange.

Approach

The purpose of *Implement Assets* as a core process within the ODM life cycle is not to prescribe a particular software engineering methodology for asset implementation. Rather, the focus of this process is to address certain activities that must be integrated into asset implementation in order to utilize previous results of domain modeling and asset base architecture, and to address certain issues that will be faced regardless of what implementation techniques are used. The Implement Assets task can be viewed as a set of guidelines for applying the supporting methods selected by the organization to implement the assets.

With relation to the completed ASSETS, the ASSET SPECIFICATIONS take on more benefits. They can enforce the specific degree of information hiding judged appropriate by the asset implementor, so that the implementation of the ASSETS can evolve over the lifetime of the ASSET BASE. At the same time, they can provide much more explicit detail about run-time performance, optimization criteria, etc., than a typical component specification.

Workproducts

■ ASSETS

Each ASSET contains the following information:

- An *asset interface specification*, expressed in terms of domain-specific features as documented in the DOMAIN MODEL. The interface specification explicitly documents the intended *scope of applicability* for the ASSET, the range of application contexts for which it has been designed.

- An *asset test and validation plan*, derived from the overall TEST AND VALIDATION PLAN. This plan provides supplemental documentation of the contextual assumptions and semantics of operation for a single ASSET.

- One or more *asset implementations* that satisfy the asset interface specification. These can take the form of a software component, a parameterized template, a generative tool that produces an asset instance at the point of demand, or any of a variety of other forms. There may be multiple implementations per specification to support different implementation technologies or features within different development settings.

- An *asset delivery mechanism* which provides any needed support for delivering the needed ASSET instantiation to the customer at the point of demand. This can include documentation of suggested integration, validation, and tailoring strategies.

- *Asset supporting material* to provide additional forms of support to application engineers in reusing the ASSET. For example, this could include example applications or a list of potential customer sites derived from the ASSET BASE DOSSIER.

When to Start

- Implementation technologies have been selected to apply to each of the ASSET SPECIFICA-TIONS, and the necessary infrastructure (technical, organizational, and educational) for the selected technologies is in place. Heterogenous technologies can be applied.

- Architectural constraints on the assets to be created have been specified (e.g., loose versus strong coupling to other assets).

Inputs

- ASSET SPECIFICATIONS. These are the primary inputs to asset implementors. The specifications allow implementors to work reasonably independently, since architectural considerations have been factored into the development of the specifications.

- EXEMPLAR SYSTEM ARTIFACTS. Candidate artifacts for reengineering into ASSETS.

Controls

- ASSET IMPLEMENTATION PLAN. Details of the technology to be used and the implementation schedule for each ASSET.

- TEST AND VALIDATION PLAN. A plan that describes how to test and validate individual ASSETS. This guides the functional testing of each asset as well as evaluation of its "fitness" relative to prescribed architectural constraints and feature interdependencies.

Activities

➤ Develop interface specification

Working from the individual ASSET SPECIFICATIONS, determine the degree of information hiding appropriate for *asset utilizers* and transform the specification accordingly. The specification should include any instructions regarding application in various settings, integration and adaptation techniques, testing strategies, etc. Record the results in an *asset interface specification*. A separate interface level may be required for developers of other assets who will make internal use of the ASSET within the asset base.

Depending on the implementation technology selected, each feature-oriented ASSET SPECIFICA-TION may be elaborated into a set of conventional component interface specifications. One ASSET SPECIFICATION may be implemented by a single highly parameterized component, a set of component variants, or a generative system. If technologies applied evolve over time, these specifications will change but the original ASSET SPECIFICATION should remain fairly stable.

Just as some ASSET SPECIFICATIONS will not be developed further in various implementation stages, it is also possible to refine an ASSET SPECIFICATION into its asset interface specification without completing the implementation.

➤ Implement each asset

Apply the technology specified in the ASSET IMPLEMENTATION PLAN to produce the *asset implementation* (or implementations, if appropriate). This activity is essentially the interface to a large repertoire of techniques that we have classified into the broad supporting method areas of Component Development, Generator Development, Reengineering, and System Modeling identified in Section 8.0.

A primary success criterion for the quality of the domain engineering process that leads to these activities will be the extent to which the models and other workproducts created serve the asset implementor in making effective use of this repertoire of techniques.

➤ Identify resources to support the implementation effort

For each ASSET to be produced, systematically consider the resources to aid in the task that have resulted from previous tasks in the domain engineering life cycle. Consider the following types of resources:

- EXEMPLAR SYSTEM ARTIFACTS examined during the *Model Domain* phase. Since existing artifacts will only loosely match the required feature profile, there may be several candidates. Where this would be a disadvantage for an asset utilizer, the asset implementor can treat these multiple artifacts as pre-existing asset prototypes that serve as contrasting templates or examples to help make alternative design choices apparent. (See the Guidelines below for further elaboration.)

- Other assets within the evolving asset base. The ASSET BASE ARCHITECTURE specifies which assets *must* be tightly or loosely coupled (to support various asset ensembles and configurations) and which must *not* be so coupled (to support separate selectability). Between these two constraints there are possibilities for leveraging assets, interim versions, supporting tools, etc., from the asset base development effort as a whole.

- Assets drawn from existing asset bases. This looks forward to an assumed scenario when multiple domain engineering efforts may have been performed or may be performed concurrently, perhaps in closely bordering technical areas. This scenario introduces constraints on the process that have been reflected in the care taken in the *Define Domain* sub-phase of *Plan Domain*. Here, the scenario presents opportunities as well as constraints.

➤ Validate assets

Refine and encapsulate elements of the overall TEST AND VALIDATION PLAN to an *asset test and validation plan* specific to each ASSET as implemented. Validate that:

- The assets respect the internal and external architectural constraints specified in the ASSET BASE ARCHITECTURE.

- Each asset implementation meets its specification. A particular point of concern here is that assets have not been overbuilt, and there are no unanticipated overlaps or conflicts among assets.

For further validation, integrate the ASSET into at least two applications within its intended scope of applicability. It is necessary to test not only the functionality of the ASSET but also that it fulfills its mission in terms of support for variability.

➤ Develop asset delivery mechanisms and supporting material

Produce *asset delivery mechanisms* and *asset supporting material* which offer support to application engineers in reusing each ASSET. These can include:

- Documentation of suggested integration, validation, and tailoring strategies.

- Examples of how to apply the ASSET, or a list of customers who have applied it successfully.

- Documentation of all legacy artifacts consulted in engineering the ASSET. Depending on the closeness of match in the feature profile, the relevant legacy systems are now candidate systems into which to back-fill the engineered ASSET. This information may be used by application engineers directly or be used for asset marketing activities in Asset Base Management (beyond the scope of domain engineering).

When to Stop

- A set of ASSETS have been completed that provide adequate coverage for a specified stage of the ASSET IMPLEMENTATION PLAN.

- Each ASSET has been implemented utilizing the selected technology and in accordance with the ASSET SPECIFICATION.

Guidelines

- Consider the spectrum of reengineering options. Artifacts identified as pre-existing prototypes will typically require reengineering to conform to the ASSET SPECIFICATIONS. Since assets may be drawn from any phase of the life cycle, asset engineering can at times resemble forward engineering, at times reverse and/or reengineering.

  Example. If both reusable designs and reusable code assets are to be developed, then reusable code components could be developed by *reengineering* legacy code artifacts, by *forward engineering* reusable design artifacts or assets into code assets, or some combination of these strategies. Alternately, design assets could be derived by *reengineering* legacy designs, from *reverse engineering* code artifacts or assets into designs, or some combination of these strategies.

  Traceability information needs to have been maintained to locate artifacts with feature profiles close enough to the asset feature profile. Whether used as prototypes or not, each identified match with legacy artifacts provides information useful in later marketing of assets.

  Caveat. In accessing artifacts in this way, the asset implementor is essentially engaging in a variant of *ad hoc reuse*, since the artifacts being reused have not been reengineered for reusability. The rationale, of course, is that asset implementors in the domain engineering context do this *once*, thereby producing the very reengineered assets required for more systematic reuse on an ongoing basis. But the implementor should bear in mind that at this particular juncture, the domain engineering process is exposed afresh to *all the risk factors in ad hoc reuse*.

  Support technologies available to support *ad hoc* reuse (textual search, data mining, etc.) may

be applicable to support this process.

- <u>Bootstrap from assets built in earlier implementation stages</u>. This will help achieve the greatest synergy and economy of joint development possible for the ASSETS. It also serves as a form of validation for the ASSET SPECIFICATIONS and the ASSET BASE INFRASTRUCTURE as it evolves.

- <u>Stick to the asset specification; don't "overbuild"</u>. The most important guideline in this part of the process is, in some ways, counter-intuitive: In implementing the asset, it is critical to stick precisely to the specification developed from the prescriptive feature model.

  In non-reuse based system development, there is rarely any justification for expending additional resources to "over-build": i.e., to implement a component in a way that exceeds specifications or requirements for the projected usage settings. *Ad hoc* approaches to reuse may circumvent this policy in a number of ways. Implementors may expend "hidden" labor to add additional functionality (i.e., "make the component more general/reusable/flexible"). The temptation will be stronger if it appears that such additional functionality will not disturb the performance of the overall system. Only in special applications like mission-critical software that must be formally verified will problems like "dead code" (within a given system context) or non-utilized functionality be of concern.

  Such overbuilding could be disastrous within an ODM domain engineering process. The whole point of the domain modeling and architecture effort has been to design the "envelope of variability" for an individual asset within the context of a well-structured asset base. Expanding the functionality of one component will change this overall design; this is the *wrong place* to make that decision.

- <u>Don't optimize merely out of habit</u>. On a related issue, many developers grow accustomed to making design decisions based on assumed priorities about which engineering qualities to optimize, e.g., performance. In a reuse context, components may be grouped to satisfy various feature ensembles designed to optimize for different priorities; e.g., space versus time efficiency. The implementor should respect these priorities. This may be an unusual way to design software, but will make sense in the context of the ASSET BASE.

- <u>Validate general reuse guidelines against domain-specific constraints</u>. Many guidelines recommend building reusable components to be extremely robust in the face of invalid data. However, in an asset base it may be desirable to have a "lean" version of a component that makes strong assumptions about its usage settings and thereby achieves a strategically important level of performance. This may appear to fly in the face of general design-for-reuse guidelines, but may be appropriate in a domain-specific context. In ODM, these general guidelines must be reviewed carefully for built-in assumptions that may conflict with the ASSET BASE design.

7.3.3 Implement (Asset Base) Infrastructure

The primary focus of this process to is implement mechanisms and procedures that provide application engineers with access to coordinated groups of ASSETS satisfying specific feature ensembles. The main thread of activity is transforming the ASSET BASE ARCHITECTURE into an implemented domain-specific infrastructure. In addition, the DOMAIN MODEL (which can be viewed as the language formed of domain-specific features) leads to domain-specific extensions to the general asset base infrastructure, beyond the scope of any one asset, but particular to the DOMAIN MODEL upon which the ASSET BASE is founded.
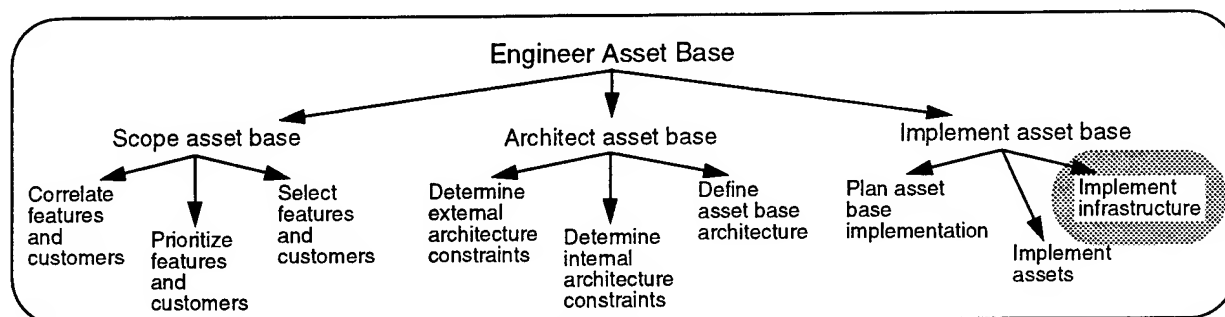
Exhibit 95. Implement Infrastructure Process Tree

Approach

The ASSET BASE INFRASTRUCTURE addresses the functionality required to assist application developers (asset utilizers) in:

• Being aware of the availability of an ASSET BASE covering a domain of interest;

• Searching the ASSET BASE for the ASSET(S) that match their system needs;

• Retrieving, adapting and integrating the ASSET(S) into their development process and/or products;

• Contributing any appropriate feedback to the ASSET BASE with respect to their experience with the ASSETS.

This process specifically addresses those elements of the ASSET BASE not confined to particular ASSETS, and requiring domain-specific extensions to any general-purpose technology utilized for the ASSET BASE. It also addresses implementation tasks most appropriately performed by or in coordination with asset developers rather than as part of ongoing maintenance of the ASSET BASE. Other aspects of ASSET BASE INFRASTRUCTURE will be ongoing maintenance and sustaining activities that will be the responsibility of an Asset Base Management organization. (In CFRP terms, these activities would fall under Asset Management rather than Asset Creation.)

As with *Implement Assets*, the *Implement (Asset Base) Infrastructure* task can be viewed as a set of guiding principles for applying selected supporting methods and technologies in a way that is consistent with the ODM philosophy and the domain engineering workproducts produced to this point.

Note that although the ASSET BASE INFRASTRUCTURE will likely impose some constraints on the application engineering processes employed when using the ASSET BASE, the intent here is not to necessarily impose a one-size-fits-all application engineering process on ASSET BASE users. In general, the infrastructure should be adaptable to a range of reuse-based engineering processes. However, depending on the perceived needs of asset base customers, it may be desirable to build the infrastructure in a way that supports a very specific application engineering approach. In fact, it is possible to define the supported application engineering process as part of the ASSET BASE INFRASTRUCTURE, if desired. This task description does not directly address that aspect of infrastructure development.

Workproducts

■ ASSET BASE INFRASTRUCTURE

Any portion of the environments, tools, documentation, and procedures required for ongoing integrated management of the ASSET BASE that are most effectively developed in tandem with the specific ASSETS of the ASSET BASE. These may include:

- Domain-specific extensions to general infrastructure mechanisms and procedures selected.

- Technology-specific extensions to the ASSET BASE ARCHITECTURE, reflecting additional constraints and semantics imposed by technology choices on the architecture.

- Inputs to a technology transfer plan. The technology transfer plan is developed outside the domain engineering life cycle, in Asset Base Management. Asset Base Management receives and transitions the ASSET BASE to asset utilizers.

When to Start

For retrieval infrastructure:

- General-purpose asset base infrastructure technology has been selected (a tailoring and infrastructure planning process).

- The DOMAIN MODEL has been completed.

- The ASSET BASE ARCHITECTURE need not be complete.

For architectural composition:

- The ASSET BASE ARCHITECTURE is complete. Although only a subset of ASSETS may get implemented within the project scope, it is important that the composition mechanisms in the ASSET BASE INFRASTRUCTURE be robust enough to grow towards complete implementation of the elements of the architecture.

Inputs

- DOMAIN MODEL. Features from the DOMAIN MODEL form the basis for the query/retrieval interface to the ASSET BASE. This interface need not be constrained to include only *prescriptive* features, because we may want asset base customers to be able to request feature profiles that are not (yet) included in the ASSET BASE.

  Some of these may be allowed for in the architecture but not yet implemented; some may be satisfiable by ASSETS in the ASSET BASE via manual methods but without direct access or automated support via the infrastructure. Over time, if requests for a particular feature profile are frequent, the ASSET BASE may be evolved to include them or provide more automated support for them. The requests may also "stretch" the architecture in some ways.

  The requests may also be satisfied by a different asset base drawn from the same feature model. Coordination would be desirable; but a common feature language for interface should aid in this effort.

- ASSET BASE ARCHITECTURE. Used to guide the implementation of mechanisms that support interactions and interdependencies among individual ASSETS.

349

- ASSET BASE MODEL. Describes the intended customer settings for the ASSET BASE. This is input to the processes for communicating the existence of the ASSET BASE to potential customers, and for designing the retrieval infrastructure.

Controls

- INFRASTRUCTURE IMPLEMENTATION PLAN. The INFRASTRUCTURE IMPLEMENTATION PLAN includes technology selections and scheduling constraints for infrastructure development relative to development of ASSET releases. Used to guide the detailed performance of the task.

- TEST AND VALIDATION PLAN. Since the infrastructure will support separate configurations of ASSETS, it may need to support incremental releases that will be used during testing and validation but not preserved as separately selectable configurations by end utilizers of the ASSET BASE.

Activities

Many aspects of ASSET BASE INFRASTRUCTURE development will rely on general-purpose (i.e., non-domain-specific) representation systems and tools. Selection, acquisition, and/or development of such technology are considered within the domain of supporting methods in the area of Asset Base Infrastructure Development. There are also dependencies with other supporting methods. described in the paragraphs below. See the appropriate sections in Section 8.0. The following activity descriptions focus on the main *process* aspects of this task.

➤ Extend base infrastructure technology

General asset base infrastructure technology will usually need tailoring and/or adaptation to requirements for the specific domain. Working from any infrastructure selection decisions incorporated in the INFRASTRUCTURE IMPLEMENTATION PLAN (some may be made in this task), and infrastructure requirements stated or implied in the ASSET BASE ARCHITECTURE, ASSET BASE MODEL, and the original DOMAIN MODEL, make the domain-specific extensions required.

Making these extensions may involve some integration with the Taxonomic and/or Concept Modeling supporting methods employed during the *Model Domain* phase, or the architectural representations selected for structuring the ASSET BASE in the *Architect Asset Base* sub-phase. One or both of these kinds of representations will likely form a strong basis for the ASSET BASE INFRASTRUCTURE because retrieval methods should be closely related to the DOMAIN MODEL and compositional methods should be closely related to the ASSET BASE ARCHITECTURE. Note that these two representations may themselves be closely related since the latter was derived from the former.

➤ Implement retrieval and qualification support mechanisms

Provide support for the following:

- *Retrieval* of ASSETS based on requests from utilizers expressed in terms of the features of the DOMAIN MODEL and/or ASSET BASE MODEL. This could include support for queries, browsing, navigation, and guidance in selecting ASSETS.

- *Qualification* of ASSETS with respect to their specifications and placement in the ASSET BASE.

Also consider additional requirements, such as support for *interchange* of assets with other asset bases, as the technology matures.

➤ Implement architecture and composition mechanisms

Working from the ASSET BASE ARCHITECTURE, implement support mechanisms that will allow separate access to ASSETS satisfying any of the specified feature ensembles. These may include any or all of the following aspects:

- Specification/selection of multiple sets of ASSETS;

- Delivery of configurations, with the appropriate asset-to-asset interfaces, glue code, etc. (The latter may need to be generated on demand, e.,g., if combinatorial configurations are possible.)

- Development mechanisms to verify that dependencies have not been engineered into ASSETS that violate the architectural constraints imposed.

- Configuration management support for ongoing tracking and management of various configurations as required.

- Point-of-utilization configuration of sets of ASSETS specified as separately selectable in the ASSET BASE ARCHITECTURE.

  Example. Simple examples of such mechanisms are the installation procedures provided with most large commercial software packages. These have evolved into separate support applications in their own right that seamlessly handle many of the chores of installation and the complexities of variable platforms and configurations for customers. Each usually provides a set of options such as "minimum", "standard", "full", and "custom" installations; for the latter, users are walked through the choices of what modules to load. Note that there are usually constraints on what modules require the presence of other modules to execute; and there is also usually a sizable core module that cannot be customized by the user. Thus such support tools reflect underlying architecture choices made for the product.

Consider use of Generator Development supporting methods for the infrastructure as well as for individual assets; e.g., for smart configuration management, making inclusion of required components relatively transparent to asset utilizers.

➤ Validate asset base infrastructure

Basic validation measures include:

- Formally verify that the set of ASSETS supporting each feature ensemble specified in the ASSET BASE ARCHITECTURE can be separately obtained from the ASSET BASE.

- Verify experimentally that sample customers from the intended ASSET BASE market can successfully retrieve, adapt and integrate appropriate ASSETS as necessary using the infrastructure.

More extensive validation could include the following:

- An asset utilizer specifies a feature profile for an ASSET within the scope of the domain, but not implemented in the (current) ASSET BASE.

- Asset utilizers requesting a feature profile outside the domain scope are effectively informed

351

that the request is out of scope and do not conduct a wasted search.

- A different *Engineer Asset Base* initiative, based on the same DOMAIN MODEL, is able to reuse much of the retrieval infrastructure developed for this ASSET BASE.

➤ Support technology transfer planning

Using the ASSET BASE MODEL, specify any appropriate aspects of strategies for communicating the availability of specific ASSETS and the ASSET BASE as a whole to the relevant application groups. Address aspects requiring access to detailed knowledge about the implementation strategies of the ASSETS.

When to Stop

- The infrastructure has been integrated with an initial set of ASSETS.

- The retrieval infrastructure and other asset utilization support mechanisms are implemented and have been used by the asset implementation team.

Guidelines

- Align with organization's software engineering practices. As with ASSETS themselves, implement the ASSET BASE INFRASTRUCTURE using methods that are consistent with the organization's software engineering methods.

- Don't co-opt asset management tasks. Don't take on tasks more appropriately performed by those people who will manage the ASSET BASE on an ongoing basis. The ASSET BASE MODEL provides a reasonable basis for developing a technology transfer plan for promoting the ASSET BASE. However, since these activities will be ongoing they are more appropriately performed, hence planned, by Asset Base Management staff.

- Let asset implementors use the retrieval infrastructure to find previously built ASSETS from earlier phases and to check interactions with ASSETS being built by other developers. This serves as early validation of the infrastructure.

- Be incremental about optimization. There is a big difference between providing *all* and providing *only* the functionality required for a given feature ensemble. Consider staging the infrastructure development if necessary. This staging strategy will be distinct from the staging of the development of the ASSETS themselves.

Part III: Tailoring and Applying ODM

Part II presents a detailed description of the core ODM process model. The model offers considerable guidance for performing the core ODM life cycle activities:

- selecting and defining a domain of focus

- modeling the range of potential variability within the scope of the selected domain

- engineering an asset base that satisfies some subset of the domain variability, based on the needs of specific target customers

Although ODM encompasses all of domain engineering in this way, the core method offers prescriptive and detailed guidance only within a relatively narrow scope, focusing on activities that are unique to *domain* engineering and for which ODM offers a unique or distinctive approach. Other activities that are under the umbrella of the ODM life cycle because they are needed for domain engineering, but not unique to it, or for which there are a number of valid methodological options from which to choose, are relegated to the category of *supporting methods*.

The scope of the core ODM method is further constrained to primarily address activities that are *necessary* for each of the major life cycle activities above. A number of value-added capabilities could be incorporated into ODM to address particular needs, but some organizations that do not have those needs could find the extra capabilities burdensome. Thus, ODM relegates such capabilities into a set of process *layers* that can optionally be selected by organizations that require them. Exhibit 96 shows how ODM has been designed in terms of these layers and supporting methods. (This exhibit was included in the Part II introduction but is repeated here for convenience).
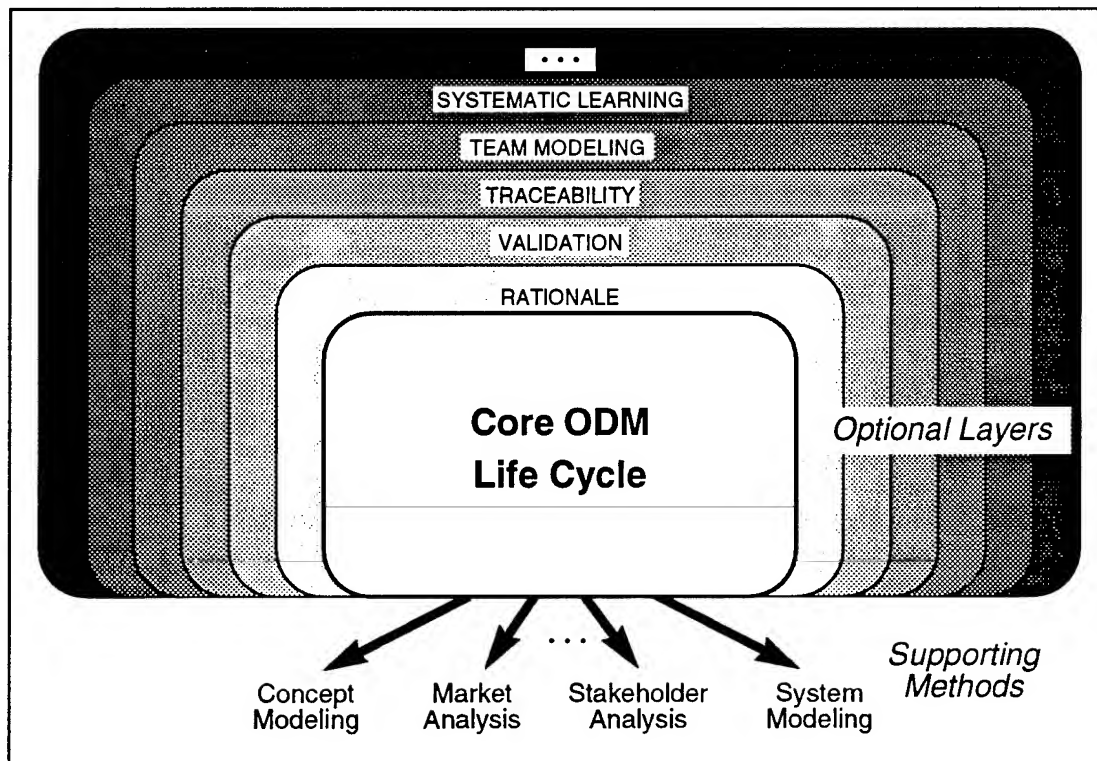
Exhibit 96. ODM Layers and Supporting Methods

The advantage of designing ODM in this way is that it enables clear definition of the boundaries and interface points between ODM and other methods and techniques to make ODM as tailorable and adaptable to differing project and organization contexts as possible. The burden that accompanies this advantage is that the tailoring activity itself can be a substantial challenge. Some of the specific objectives (and, viewed in another way, challenges) of the tailoring process include:

- Defining the role of ODM within a specific overall software engineering process or life cycle.

- Configuring the specific domain engineering processes to be performed and their interactions.

- Selecting and integrating the specific set of supporting methods that will be most effective and applicable within the context of the projects, organizations, and domains in which ODM is applied.

- Similarly, selecting and integrating the added layers of capability that are needed across ODM to meet project or organization objectives.

- Selecting and integrating the tools that are needed to support the selected methods and layers, as well as the core ODM techniques.

The purpose of Part III is to offer some general guidance in how to tailor and apply ODM for use in specific domain engineering projects.

- Section 8 provides brief descriptions of a number of categories of supporting methods and can thus assist in identifying and selecting specific methods.

- Section 9 plays a similar role for the optional ODM layers.

- Section 10 offers significant practical guidance for tailoring and applying ODM, ranging from guidelines for staffing and budgeting domain engineering projects to suggested strategies (with examples) for how to address some of the tailoring issues noted above. This section offers a number of practical hints and suggestions reflecting the experiences and lessons learned from past and current applications of ODM.

Unfortunately, in this version of the guidebook, the information provided in these sections is incomplete relative to what was planned. Treatment of supporting methods and layers is uneven; significant useful information is included for some methods and layers, while little is said about others. The guidelines offered in Section 10 are similarly uneven. These imbalances reflect only on the amount of time that was available to write the guidebook, rather than on the relative importance of particular methods, layers, or areas of guidance.

Detailed and extensive ODM tailoring guidance is beyond the scope of the guidebook. In particular, the issue of automated tool support for ODM will not be addressed here. In this and other areas, supplementary guidance may appear in separate documents in the future.

8.0 Supporting Methods

In this section, we briefly discuss aspects of supporting methods in relation to their role in ODM-based domain engineering. As discussed in the introduction to Part III above, the treatment of specific supporting methods in this section is uneven. We have at this time elected to document in some detail the supporting method areas we believe are least likely to be familiar to a well-trained software engineer, the primary audience for this document. Most of the software engineering specific supporting methods, like reverse engineering and system modeling techniques, are quite thoroughly covered in other sources.

What is a Supporting Method?

In the ODM context, we encapsulate as a supporting method any body of techniques, methods, and tools critical to the accomplishment of overall domain engineering project goals, but

- not unique to domain engineering; and/or

- highly dependent on the specific needs of each organization applying ODM.

Applicable methods used in other kinds of engineering tasks, or other disciplines entirely, are usually well-documented elsewhere. ODM's interactions can be considered as "subroutine calls" to these well-defined external disciplines. Other supporting methods involve sets of specialized techniques highly relevant and specific to domain engineering but varying according to the organization, the domain, or the representations and tools selected. Documentation of such methods in sufficient detail is beyond the scope of this document. A supporting method may include any or all of the following elements:

- a modeling representation language and/or notation;

- a specific set of methods, skills and competencies;

- a repertoire of best-practice or recommended options for the area of concern; and

- specific tools, technologies, training materials, specialists, and a technical literature.

Selecting appropriate supporting methods will be one of the primary infrastructure planning tasks of the domain engineering project planner. There are usually many approaches possible for each type of method, and the factors to be considered in selection of the methods will usually extend well beyond the scope of the domain engineering project. The intent in this section is to list a set of supporting method areas relevant to ODM, provide brief descriptions of selected supporting method areas, and identify particular constraints or considerations relevant to the role of these methods within ODM. The purpose is not to make general recommendations or comparisons.

Supporting Method Areas

Though there are varying reasons for separating supporting methods from the core ODM life cycle, the interface to supporting methods in ODM has been structured in a consistent way. This section clusters supporting methods into a set of broad categories or areas that represent an initial attempt to identify the supporting methods most essential for ensuring the overall success and quality of an ODM project. The categories are provisional only; there may be significant overlaps between them, and there may be useful supporting methods needed for ODM that are not included here. The ODM process model and method are not intrinsically dependent on this particular cate-

gorization, although for convenience and clarity the process descriptions in Part II are written to reference the supporting method areas identified here. These supporting method areas are:

- Stakeholder Analysis

- Information Acquisition

- System Modeling

- Process Modeling

- Reverse Engineering

- Concept Modeling

- Lexical Analysis

- Market Analysis

- Reengineering

- Component Development

- Generator Development

- Asset Base Infrastructure Development

Of these, Stakeholder Analysis, Information Acquisition, Concept Modeling, and Market Analysis are described in significant detail in the following subsections. These descriptions begin with an overview of the supporting method area, including discussion of the general problems and tasks it addresses. A brief survey/typology of the range of options and/or a set of illustrative examples may be provided. These are not exhaustive catalogues nor prescriptive sets of recommended choices. Following the initial overview, the following topics are discussed:

- *Role in ODM*: The general relationship between the focus of the supporting method and the overall goals of domain engineering; and the specific tasks within the ODM life cycle where interaction with this supporting method is of most importance. How critical is the supporting method to ODM? There are four basic reasons for considering a given supporting method.

  — *Must Do*: If this supporting area remains unaddressed your project is at risk. For example, taxonomic modeling techniques are intrinsic to domain engineering. If you do not explicitly select a method you will still be performing these activities, but probably in an *ad hoc* way. These are supporting methods because the specific representations used, the degree of formality involved, etc. will be project-specific.

  — *Nice to do if you can*: The method area may not be appropriate for all projects, depending on their scope and resources. Access to personnel with these skills would generally enhance a domain engineering team. Results of work in these areas might prove useful data for domain engineering; similarly, domain engineering results could be useful to work in the supporting method area.

  For example, generative techniques are an important part of the implementation repertoire for reuse, and the ODM method helps in discovering opportunities for applying these techniques. But it is quite possible to build a well-structured asset base that will be utilized effectively that does not employ generative techniques. The Army/STARS Demonstration project was organized at a high level as a coordination between a domain

engineering and a system reengineering team, with independent but mutually supporting objectives [ADER96a].

— *Synergy possible*: For example, suppose you are examining legacy designs during the *Acquire Domain Information* sub-phase and you only have access to code for some of the representative systems. If there are other reasons for the organization to do reverse engineering of those systems (e.g., to bring legacy systems into line with company standards for design documentation) then these tasks may dovetail cost-effectively with domain engineering.

— *Conflict Possible*: If you're performing activities related to this supporting method you need to be in coordination; e.g., a possible overlap of a domain engineering effort with a process improvement campaign in the company.

- *Interface*. A description of the information going into the method in the ODM context and the information expected to come out (these can be thought of as input and output parameters, in high level terms). For some methods, this may be difficult to characterize precisely and will have to be negotiated on a case-by-case basis.

- *Guidelines*. Constraints imposed by ODM on selection of specific methods, and guidelines on integration of the methods and techniques with the domain engineering processes described in this document.

Mapping Supporting Method Areas to the ODM Life Cycle

Calls to the methods listed above appearing in the process descriptions in Part II of this document include the following (arranged in the approximate order in which they are first encountered in the ODM life cycle):

- *Stakeholder analysis* techniques are used throughout the *Plan Domain* phase, but particularly in the initial sub-phase, *Set Project Objectives*.

- The subsequent sub-phases, *Scope Domain* and *Define Domain*, represent aspects of the life cycle fairly specific to domain engineering; no specific supporting methods are included for these sub-phases. However, because of the strategic importance of domain scoping, and the methodological importance of domain definition, success in these activities will depend to a large extent on integrating the organizational and engineering disciplines represented by the supporting methods as a whole.

- *System modeling, reverse engineering*, and *process modeling* are all used in the *Acquire Domain Information* sub-phase of the *Model Domain* phase. These are techniques familiar to most software engineers. In addition, a repertoire of *information acquisition* techniques can be used in the *Acquire Domain Information* sub-phase that draw on a variety of non-software engineering disciplines. *System modeling* is also used in the *Engineer Asset Base* phase, both to support the *Architect Asset Base* sub-phase and the *Implement Assets* task.

- *Lexical analysis* and *concept modeling* techniques are used in the *Describe Domain* sub-phase, in the *Develop Lexicon* and *Model Concepts* tasks respectively. *Taxonomic modeling* techniques are special kinds of concept modeling techniques called out specifically in some cases. A variety of techniques can be used in the *Model Features* task, ranging from informal representations to more formal taxonomic techniques. The distinctions between these areas are discussed in Section 8.3.

- *Market analysis* techniques are used in the *Scope Asset Base* sub-phase of the *Engineer Asset Base* phase of domain engineering. They play an analogous role here to the stakeholder anal-

357

ysis employed at the start of *Plan Domain* and the information acquisition techniques employed at the start of *Model Domain*. Stakeholder analysis focuses on the full range of stakeholder roles, information acquisition on informant roles, and market analysis on customer roles.

- A group of supporting methods is called out to address asset implementation. In addition to *system modeling* mentioned above, these include *reengineering* techniques for transforming artifacts into assets, *component development* techniques for implementing software components designed explicitly for reuse, and *generator development* techniques for applying transformational and generative technologies to asset implementation. These are all applied primarily in the *Implement Assets* task.

- The *Implement Infrastructure* task employs *asset base infrastructure development* methods focusing on mechanisms for supporting integrated management, retrieval and evolution of assets. Examples of this technology would include reuse library mechanisms, Web-based infrastructure, and program composition tools.

In addition to the supporting method areas listed above, there are at least three other aspects of the ODM method that could readily incorporate supporting methods. However, these areas are still emerging as technical disciplines (in general, they are research topics) and there are few if any specific methods available to fill these ODM roles. Applicable techniques are currently described within specific ODM process descriptions in Part II. We envision that these techniques will become well-elaborated enough in the future to warrant packaging them as distinct supporting method areas. These areas include:

- *Innovation modeling* is a critical aspect of the *Refine Domain Model* sub-phase, particularly in the *Resolve Domain Model* task. Although there are many general innovation techniques, most would not be generally applicable to the core domain engineering life cycle without considerable integration.

- *Model integration* as it relates specifically to the integration of ODM descriptive models is the focus of the *Integrate Descriptive Models* task in *Refine Domain Model*.

- *Asset base architecting*, relevant to the Architect Asset Base sub-phase of *Engineer Asset Base*, involves more than just conventional system modeling or architecting techniques, because of the fact that it must accommodate potentially high levels of variability among target application architectures.

Interdependencies Among Methods

Each of these supporting method areas may have interdependencies with core ODM processes and workproducts including:

— results needed from supporting method activities to perform core ODM tasks;

— ODM workproducts that can aid in the process of selecting an appropriate supporting method in a given area; and

— workproducts ODM contributes as inputs to supporting method activities.

In addition, there may be interdependencies *between* supporting methods that need to be considered in planning the domain engineering project as a whole. For example, while in theory the system development methodology used to implement software assets in the *Engineer Asset Base* phase can be selected independently from the methodology used to reverse engineer legacy arti-

facts into a common format for comparative modeling, there are probably pragmatic reasons to make sure a common methodology is used in these two activities. In some cases, examining the overlap between two supporting method areas helps to underscore particular challenges presented in domain engineering. The following paragraphs briefly discuss one such case, the potential overlap between system modeling and process modeling techniques.

Software developers create software applications in development settings; end-users execute these applications in performing work in usage settings. Typically, system modeling techniques have been used to represent data and processes in the system's operational environment (the usage setting). Process modeling largely evolved out of system modeling techniques, but has more recently been applied to modeling the tasks of the software developers' environment itself (the development setting). In principle the same representations could be used to model both kinds of environment, but in practice this is rarely done. Also, often different people are concerned with these separate modeling tasks for different reasons: e.g., system modeling to build applications; process modeling to do process improvement.

Yet many interesting and potentially useful opportunities in domain engineering lie in the linkage between these two environments. Opportunities for reuse of both software products (e.g., components in the traditional sense) and software processes (e.g., tools that automate domain-specific software development tasks) become visible only when working with an integrated picture of both environments in interaction. Neither system modeling nor process modeling techniques are oriented towards representation of both these environments in an integrated way. One particular difficulty lies in representing the actual software applications produced, which appear as a kind of output in the developers' environment, but serve to encapsulate processes when executing in the end-users' environment.

Thus both system modeling and process modeling are listed as supporting methods to ODM, and the method can be adapted (with some constraints) to a wide variety of specific representations and methods in either of these areas. But the quality of domain engineering results will increase to the degree that the methods selected can be smoothly integrated (at best, if a single method is used in both areas). In addition, extensions to the state of practice in both method areas will be needed in order to obtain the full benefits of an ODM-based approach to domain engineering.

This is one extended example of the many inter-dependencies between the supporting methods area. Further relationships of this kind may be discussed briefly in some of the following sections. A more thorough explanation would need to be part of a full guide to ODM tailoring and project planning, which is beyond the scope of this document.

8.1 Stakeholder Analysis

Stakeholder analysis techniques are employed in situations where it is important to identify all the people who are potentially affected by a particular issue or decision. Typical kinds of situations where these techniques can be useful might include:

- change management in large organizations, where mandates from official lines of authority are not sufficient to guarantee that change will be adopted and sustained;

- product-oriented companies seeking to move beyond a traditional focus on vendor/customer relations to explore broader value-chain relations and alliances;

- issues requiring coordinated actions or decisions by multiple individuals or organizations where there is no central, hierarchical control or authority (e.g., negotiation, conflict resolution, political decisions, etc.)

Though there are many techniques and approaches, most stakeholder analysis techniques share the purpose of exploring more sophisticated ways of establishing relations with a main producing or initiating organization. A key task is revealing "hidden" stakeholders, who may provide valuable information from unique perspectives, and who may be affected by decisions in ways that would otherwise not be adequately considered. Thus many methods and practitioners strong emphasize techniques that "re-contextualize" information and assumptions implicitly held by participants, and work out of a strongly held ethical stance that advocates social and cultural equity over support of status quo power relations in organizations. The ODM method has been strongly influenced by these approaches.

Stakeholder analysis plays a part in many broader disciplines, such as Total Quality Management (e.g., quality function deployment approach), business process re-engineering and organization re-design. One emerging movement that puts central emphasis on stakeholder analysis is Future Search Conferences, best documented in [Weis92]. Future searches are intensive conferences (usually two to three days in duration), carefully planned and structured to bring the most diverse set of stakeholders possible together for rapid progress in finding common ground and generating ideas for future action. (It is interesting to note that the term "domain" is used in this work to denote complexes of interacting stakeholder interests that involve multiple organizations and social groups [Tris83].)

Role in ODM

As discussed in the detailed descriptions in this document, domain engineering initiatives share many aspects with situations in which stakeholder analysis has been applied:

- Domain engineering, as part of a larger shift to reuse-based software development (or, more broadly, to a megaprogramming paradigm) involves significant dynamics of organizational change.

- Innovative approaches to reuse, like generative techniques or shifts of features across binding sites and settings, can help re-define and broaden the value chain of the software life cycle.

- Domain models and asset bases should eventually evolve into frameworks that support new business models, for new types of organizations. These will require techniques to identify a diverse set of stakeholders that span current organizational boundaries.

Stakeholder analysis techniques are used most intensively in the first task in the ODM life cycle, *Determine Candidate Project Stakeholders*. They will be useful again after the domain of focus is

selected, in the *Situate Domain* task, in identifying the stakeholders in the domain as distinct from the project. They will also be applicable in the *Plan Data Acquisition* task, for identifying a rich set of informants, and in the *Correlate Features and Customers* task, for identifying asset base customers. In the latter-mentioned tasks, they will overlap to some extent with information acquisition and market analysis techniques respectively.

Interface

TBD

Guidelines

- How essential are these techniques to ODM? Stakeholder analysis techniques mostly take the form of methods practiced by specialized consultants. Some input from consultants trained in these techniques early in the *Plan Domain* phase could have major impact on the strategic soundness of the project. If this is not possible, some reading about techniques will help you be aware of those tasks where stakeholder analysis is going on (whether formal or informal). Consider leveraging any resources in the company where these skills might be available.

- Selecting methods and practitioners. The most useful approaches to stakeholder analysis for ODM projects will emphasize facilitating dialogue among stakeholders, rather than mechanical or analytical decision processes [LIBR96]. The latter are appropriate for processes that are well-understood and mature; domain engineering is still a young discipline, and some flexibility is required. However, use concrete, visual tools and guidelines wherever possible for orderly representation of trade-offs among stakeholder interests -- conflict, synergy, etc. (For examples, see the templates shown in Exhibit C-1, PROJECT STAKEHOLDERS/ROLES MATRIX, and Exhibit C-3, PROJECT OBJECTIVES/STAKEHOLDERS INTERESTS MATRIX.)

- Categorize stakeholders into meaningful ways. Consider the roles and relations that matter most to your organization. Draw on work in the reuse community on reuse-specific roles. The CFRP provides one useful starting point [CFRP93b, Crep95a].

Organizational Consulting

Success in stakeholder analysis relies on a more fundamental set of skills that form part of the general discipline of organizational consulting. The parallels between organizational consulting and domain engineering are instructive. Three central concerns in domain engineering are directly analogous to classic organizational consultancy challenges:

- Presenting and underlying problems. Though usually brought in to help solve problems (i.e., something is not working) consultants have learned that the initial problem around which they are called in, the so-called presenting problem, rarely proves to be the real or underlying problem that needs to be confronted for the health of the organization. Much of the consultants value lies in the ability to discern and diagnose that underlying problem, and to articulate it to the organization in a way that engenders effective action.

  In a similar way, a domain engineering project is often organized around a domain as defined by the stakeholders in the organization. If the domain could be adequately described as a system family or product line given the organizations current structure, then it would rarely be called a domain. Use of this term flags a perception on the part of people in the organization that some new organizing structure for software engineering needs to be found. If they already knew the best way to do this, there would be no domain engineering project. But, as in consulting, the presenting domain recognized by the organization may not turn out to be

361

the best definition for the domain.

- <u>Influence without authority</u>. While having no direct-line management control, consultants must aid decision-makers in recognizing problems and must recommend solutions, even when this involves confronting managers about problems in organizational process, management style, or interactions. In order to get their recommendations accepted, the technical expertise of the consultant must be combined with a set of organizational consulting skills and competencies.

  In a similar way, domain engineering projects will often be initiated for precisely those opportunities within organizations that fall in between the boundaries of existing management structures: e.g., across established product lines, across separately-funded contract efforts, etc. To be successful, the asset base must satisfy multiple stakeholders with competing needs and values. However, reuse practice can rarely be merely mandated from above. Domain engineers must thus get their solutions, or their parts of solutions, adopted but may have no direct authority to make this happen [see Sche87b, Chapter 2].

- <u>Explicit contracting</u>. Largely because of the two issues mentioned above, a consultant's role becomes problematic unless the objectives, boundaries and expectations for the work and the consultant's role within the organization are established clearly and explicitly at the beginning. This is known as the *contracting* phase in the consulting task.

  Clear contracting is just as essential for the domain engineering project, for many of the same reasons. The ODM process model has incorporated the importance of contracting in the first sub-phase, *Set Project Objectives*. This is the motivation for making objective-setting internal and not external to the process; and this is the real challenge in the activities of that initial sub-phase.

- <u>Organizational Development</u>. Organizations grow and adapt according to predictable stages. Many theorists assign different names, stages, and key challenges to this development. How an organization chooses to work as well as its strategic choices are influences by its circumstances, its competencies, and its stage of organization development.

  The notion of organizational development is of direct relevance to domain engineering, particularly in setting project and organizational objectives. Software-intensive businesses are typically experiencing classic growth and maturing dilemmas at the point when they commit to software development process improvement and anticipate high reuse of their technical efforts. Such dilemmas bring a host of forces regarding performance management, lateral communication, cost management, and job design that can support or hamper the process and adoption of even extremely robust domain engineering results.

In light of these parallels, any domain engineering project would do well to get some training in consultancy and technology transfer principles. An excellent starting resource for getting acquainted with these issues in a practical way is Peter Block's *Flawless Consulting: A Guide to Getting Your Expertise Used* [Bloc81]. Modelers and managers should also be familiar with typical and predictable stages of organization development and how they may influence a domain engineering project life cycle [Liev80, Grei72]. At a minimum, someone with these skills should be available as a consultant and coach to the team (i.e., to the "domain engineers as consultants").

8.2 Information Acquisition

A domain engineering effort requires a great deal of information about a domain and a potentially large number of systems relevant to the domain. This information can come from a number of sources, including software artifacts and human informants. A large repertoire of methods for systematically acquiring this information is applicable. Any particular method will stress one aspect of data acquisition over others; some concentrate on the *systematic* acquisition of data, while others concentrate on how to acquire the data, e.g., by describing how to manage an informant during an interview. Some concentrate on the 'big picture' for managing a data acquisition project, while others concentrate on the details of managing the collected data. A typical project will have to draw on several methods; prepare to be flexible.

Artifact analysis in a software engineering context is closely allied to reverse engineering, and will not be discussed in further detail here. A complementary set of techniques for eliciting information from people, broadly referred to as *qualitative* methods, will be the focus of this discussion. Qualitative data gathering is central in countless technical fields and social scientific disciplines. Some related areas in software engineering where these techniques have been applied include:

- Knowledge acquisition techniques developed or adapted for expert systems or natural language-based systems development;

- Scenario-based requirements elicitation techniques;

- Work on capture of design rationale, e.g., protocol analysis of the programming or software design process.

- Ethnographic techniques, developed for observation, analysis, and interviewing.

All of these techniques are aimed at some aspect of the problem of making tacit information explicit.

Some specific forms of information-gathering techniques include:

- *Sociological or anthropological fieldwork* methods seek to develop a picture or description of the linkage between human behaviors and shared cultural beliefs and assumptions, by studying artifacts, rituals, and observable events within the community of study. Interactions of field workers with the cultural insiders help make explicit the tacit cultural knowledge that is so obvious that practitioners never really notice it any more. See Spradley on ethnographic interviewing and observation [Spra79a, Spra79b], and Schwartzman on ethnographic analysis [Schw93]

- *Linguistic analysis* attempts to capture specific lexicon and language usage within a given language community. (It is interesting to note that the term "domain analysis" also occurs in this work; here, "domain" refers to a general topic area within which specific lexicon terms are collected [Spra79a].)

- Eliciting *stories* is a technique increasingly being used in organizational settings to assess and understand organizational cultures [Boji91].

- *Clinical intake* or *evaluation interviews* attempt to tap individuals' motivations, beliefs and common behaviors from a psychological point of view. These methods of eliciting data can be separated from the diagnostic categories sometimes used to label pathologies, and have been compared to ethnographic fieldwork [Sche87a].

- *Appreciative inquiry* is based upon the assumption that you get what you pay attention to. e.g., if you focus on problems or pathology, you get problems or pathology; if you focus on what works well and is beneficial, you get positive stories about the work context. This approach is increasingly being used to assess organizational receptivity to change. It doubles as a change intervention in raising awareness about what is positive. The approach is especially useful in focus group interviews in which participants are defensive or unsure of why the modeler is observing and asking questions. [Shri90]

- *Journalistic interviewing* (as practiced a la Bill Moyers, etc.) is another approach to eliciting contextual data. This approach differs from others in that getting a story and creating interest for an audience is often more important than being true to the language and/or beliefs of the informant. This approach is not recommended, but until modelers have studied other methods in some detail, differences in technique, role, and underlying assumptions may not be apparent.

- *Phenomenological methods* attempt to create a degree of objectivity in observation and meaning-making by deliberate suspension of the field worker's agendas and intents in order to allow a phenomenon to reveal itself, rather than probing it. This is the opposite of journalistic interviewing: very non-invasive, but also not goal-directed. Though probably the hardest style to learn systematically, elements of this approach underlie all non-directive observational techniques, including the ethnographic approaches. (See, for example, [Zubo88], Appendix B, "Notes on Field-Research Methodology"; also [Wino87], Chapter 3, "Understanding and Being.")

- *Psychological methods* draw on research about human memory to help to structure tacit knowledge. A different theory of the structure of human memory can be used depending on the requirements of the particular situation. The acquired information is structured according to the selected theory of memory, and further questions are posed, based on what that theory predicts will be easily accessible. For an example of a psychological method based on personal construct psychology, see [Manc88].

- *Scenario-based engineering* is based on elicitation of scenarios of use in order to derive specifications of systems. A number of scenarios can be combined into a single scenario diagram. For an example of scenario-based engineering, see [Hadd94].

Role in ODM

These techniques are important in domain engineering because developers embed numerous tacit assumptions within software artifacts, usually unintentionally. These implicit contextual assumptions only become visible (sometimes disastrously!) when reuse is attempted in another context. In ODM, an entire sub-phase is dedicated to the acquisition of domain data.

A good domain modeler should be able to play the role of "techlorist," and be able to catalog information previously only implicitly available in the culture of the system developers and users. This requires a set of skills not usually considered part of the software engineer's discipline. Thus it is often advisable to mix capabilities on a domain modeling team. There have been highly successful projects that combined trained ethnographers and human factors researchers with software designers [Brow91, Lind93].

An unusual aspect of information acquisition in ODM is that system developers, as well as end users, are considered as information sources for a system. Most knowledge acquisition techniques were designed with end users as the planned information sources. In principle, any technique that can be used for end users can be used on system developers as well, but certain special problems can arise.

First, since domain modeling can be confused with system modeling, there is the double danger that the investigators, already familiar with system modeling, will be tempted by their own biases to force a model on the data that is not intended by the informants themselves, and that the informants will see this simply as yet another systems modeling effort. Furthermore, the informants themselves might resist a technique that appears to be too non-directive and unfocused, even when these attributes are deliberate aspects of the techniques.These problems need not cripple the data acquisition effort, as long as the modeling team takes them into account when selecting and carrying out a particular method.

The primary role of these techniques in ODM is in the *Acquire Domain Information* sub-phase. Naturally, information will be gathered from stakeholders throughout the entire process; but it is this sub-phase where being more systematic about data-gathering becomes critical.

During the *Plan Data Acquisition* task, the major role of information acquisition techniques are in determining goals for data acquisition. In this activity, the DOMAIN DOSSIER includes information that has been elicited so far. The planning job of an information acquisition technique is to determine what data needs to be elicited to complete the data set. It can draw on existing data to set goals in terms already known in the dossier, or it can draw on the structure of the elicited data to determine whether another set of data should be elicited. The method should return a goal for data acquisition, in the form of some piece or set of information that needs to be acquired.

Information acquisition techniques also play an important role in planning elicitation logistics. Logistic questions such as the size of teams, the meeting place, how the agenda is presented and managed, etc. can all be influenced by the information acquisition method. Different methods vary on what they need as input in order to plan elicitation, but in general, they need to have information about the goal(s) of the session, as well as information about the available investigators and informants. It returns any appropriate logistic considerations, such as composition of the session, location, time, equipment, preparation materials, etc.

During the *Elicit Data* task, information acquisition techniques are applied to achieve the goals set out during the *Plan Data Acquisition* task above. In particular, a goal will be given, which specifies information to be elicited from some informant, along with any relevant information about the setting in which that informant works, background information about the domain and information about biases that the observer should be aware of. During *Elicit Data*, the investigator will use the method to acquire the specified information; this will typically include special questioning types and observation methods tailored to the particular needs of the situation. The output of this activity will be the data itself, to be used during *Integrate Data*, below.

During the *Integrate Data* task, the major role of the information acquisition technique is to provide a framework into which to structure the data that has been elicited. This is the same framework that is used during planning to determine what data remains to be elicited, so there must be strong coordination between these two parts of the method. This part of the technique should take the current dossier, plus the newly elicited data, and return a filled-in structure that incorporates both the extant dossier as well as the new data.

Linguistic analysis techniques may be applicable as well in the *Develop Lexicon* task. Scenario and narrative-based methods become important in the *Interpret Domain Model* task, when modelers attempt to reconstruct rationale for commonality and variability across representative systems. Appreciative inquiry and related techniques may be useful in the *Prioritize Features and Customers* task, when customer interest in various feature combinations, including innovative features, is assessed.

Interface

The information acquisition techniques supporting method area is actually a coordinated suite of techniques applicable to various tasks of the *Acquire Domain Information* sub-phase. Each of these has its own input/output profile. These are given here for three calls to information acquisition mentioned above.

- Determine data acquisition goals:

  Input: A structured set of data; e.g., a hierarchy, table, list of answers, etc.

  Output: A request for a particular datum or set of data

- Interview informant:

  Input: A request for a particular datum or set of data
  Information about the informant; what setting he works in, terminology
  Information about the stage of the relationship between informant and investigator

  Output: The requested data

- Record data

  Input: New data to be recorded
  A structured set of data so far

  Output: A structured set of data, including both the original set as well as the new data

Guidelines

- Use the skills available to the team. Acknowledge that these tasks require special skills and assess the team honestly in terms of these skills. However, remember that not all these skills are scholarly and research-oriented. Many are practiced intuitively by successful writers, journalists, and professionals in many fields. An excellent source of talent in this area can be found in the technical writing/documentation groups of many organizations.

- Respect confidentiality. If necessary, advise informants not to share information that needs to be treated as confidential, if you can't guarantee you'll be able to do this; or if you suspect the informant is using the role as a "mouthpiece" for other agendas. This is one aspect of many complex ethical considerations

8.3 Concept Modeling

Concept modeling lies at the heart of domain modeling. It may seem a bit strange to see so central a concern treated as a supporting method. Recall that a supporting method is a method that fulfills some function that must generally be carried out in order for domain engineering to proceed, but can be carried out in many ways. If the core domain modeling process in ODM does not specify a particular set of representations and modeling formalisms to apply, then what content does it offer? How can the process be said to be independent of these representations and formalisms? How does one select the appropriate representations and formalisms, and integrate them with the core processes and other workproducts specified in the method?

If the reader has reached this point in the document, we trust they have some answer to the first of these questions. This section will address the latter two issues. In fact, we believe the separation of concept modeling techniques as a supporting method is a major strength and unique contribution of ODM, but recognize the need to explain and motivate this particular interface in more detail than many of the others. Accordingly, this supporting method is described in more detail than others within this section.

In the core ODM life cycle, descriptive modeling involves studying a set of exemplars for the domain and modeling their commonality and variability. In its purest form, this activity is generally known as *classification*. Classification has been broadly defined in [Bail94] as follows:

> "In its simplest form, classification is merely defined as the ordering of entities into groups or classes on the basis of their similarity. Statistically speaking, we generally seek to minimize within-group variance, while maximizing between-group variance... [making] groups that are as distinct (non-overlapping) as possible, with all members within a group being as alike as possible."

Even this simple definition points out some of the basic issues that must be resolved in classification. Sophisticated classification and taxonomic methods have been developed in a variety of disciplines, including the biological sciences, social sciences (e.g., survey data, sociology, ethnography, etc.), and library science, to name but a few. Each disciplinary emphasis brings certain advantages as well as certain biases and limitations to the domain modeling problem. Although we cannot offer a survey here of different techniques or guidelines for selecting the appropriate taxonomic modeling supporting method, we can explore the question: what aspects of the ODM application context present particular challenges or constraints for taxonomic methods? In this introductory section, we will discuss the applicability of taxonomic methods from two different perspectives: first, in terms of domain modeling in general; second, specifically addressing domain engineering for software reuse.

We use the term *concept model* to denote a model that describes some *set* of things or phenomena for the purposes of classification as described above. A model reflects a particular viewpoint on the things it models. Models are written in some formal language; for any particular model, we call this the *model formalism*. Generally, model formalisms are defined so that many different models can be developed that obey the structure of a common formalism. The same set of things could be described by different models in the same formalism if the things were classified differently. They could also be described by different models in different formalisms.[1]

[1.] We will defer the question of whether the "same" model could be defined in different formalisms. This requires a more formal treatment of all the definitions provided here, which is beyond the scope of this guidebook.

The kinds of relationships between things modeled that can be talked about in the models depends on the formalism. The specific relationships between things depend on the domain; the people who know the most about this are the domain informants themselves.

Elements of a concept modeling supporting method include the following:

- A *concept modeling formalism* describes a set of element types (or "concept capturing constructs") and a set of distinct semantic relationships, or structuring relation types that can be defined between these constructs. The formalism also includes constraints on configurations of the elements and relations.

- A *concept modeling representation* provides a way of documenting a specific model defined in terms of the formalism.

- A *concept starter set* provides an initial set of elements, defined via the formalism, which are generally presumed to be a useful starting point in modeling certain kinds of domains. Note that we distinguish this element from the formalism because one formalism could be the basis for many distinct starter sets. In theory, the converse is also true: there may be starter sets that are little more than lists of named concepts of interest, and hence could be "cast" into a number of different formalisms. (For simplicity, we treat these as elements of a single supporting method area, however.) Another way to look at this set is that it is a specific model "imported" into the domain modeling effort, which might then be linked with other models.

- A detailed modeling *process* for building a specific model based on terms and examples.

- Criteria for rejecting some terms from use in the model, and for introducing "slots" for new terms that are needed, based on the models developed. For example, if two terms turn out to be indistinguishable with respect to the model concepts and relations, the supporting method may return the redundant terms to the core method, with the implicit request: "These two terms are indistinguishable with respect to the model data provided. Pick one as the canonic term or provide further differentiating data."

Note that the supporting method does not try to solve the "ethnographic" problem of selecting domain vocabulary. Similarly, the supporting method does not try to come up with names for the new concept "slots" but may be able to discern that a particular slot is needed. This preserves the clarity of the hand-off between core and supporting method

> *Example.* We will use the RLF formalism as an example of a taxonomic modeling supporting method in this discussion. The Reuse Library Framework (RLF) is a domain modeling toolset developed under DARPA/STARS funding [Unis88a, Sold89a]. RLF supports a structured inheritance-based semantic network formalism originally derived from the KL-ONE family of knowledge representation systems.

In the RLF formalism, there are only a few element types: generic categories and instances or individual categories. Categories can be related via specialization links or "relationship" links; instances are related to generic categories via "instantiation" links, and instances can be linked to instances via "filler" links. Constraints in the formalism include a notion of strict (subsumption-preserving) inheritance.

As a contrasting example, consider the familiar idea of a faceted classification scheme. Such schemes generally consist of a set of facet terms, each one of which can take on a set of values. Note that there is no notion in the faceted scheme of an entity within the model that directly represents one of the set of exemplars or phenomena being classified. To describe any exemplar that is being classified, one value from each facet is selected and the resulting profile of facet values *implicitly* represents the exemplar. This is an important distinction,

because in a strict sense it implies that a faceted scheme is not really a concept model in the sense defined here.[2]

Role in ODM

- In the *Acquire Domain Information* sub-phase, there may be calls to the taxonomic supporting method if an artifact in one of the settings being investigated uses such a method. The call might be for the purpose of interpreting this artifact correctly, or the model may actually be created as part of data acquisition. It should be kept in mind that this is a different call, since the criteria for selecting the appropriate method are quite different. In this case, the main criterion is to select the method of choice within the domain setting itself. Later, in the Describe Domain sub-phase, modelers will have the chance to select the taxonomic modeling supporting method most appropriate for their own modeling efforts in light of the overall PROJECT OBJECTIVES.

- In the *Model Concepts* task, the taxonomic modeling supporting method may provide a concept seed/starter set for the CONCEPTS OF INTEREST.

- The primary call to the taxonomic modeling supporting method comes when a group of lexicon terms have been gathered together under a single concept of interest as a cover term. Also a group of exemplars is also passed to the method.

The constraints that ODM places on what kind of modeling formalism is used are a bit tricky to describe.

- In the *Model Concepts* task multiple CONCEPTS OF INTEREST and other terms are passed to the supporting method, which determines how to partition these terms into separate models. In theory there could be several models developed, using several different formalisms. Some terms might be modeled in multiple models; for example, a "native" model drawn directly from a domain informant might reflect one way of classifying the data, while another model might reflect the choices of the modeling team.

- Must there be more than one model developed?

- Must the models be taxonomic? At one time it seemed clear that all the concept models would be taxonomic in nature. At a later point it seemed that no such restrictions should apply. Models can include arbitrary "domain-native" sets of relationships or could be supporting method specific. We now believe (on a still tentative basis) that the following claim could be made:

  Unless at least one of the DESCRIPTIVE MODELS utilizes a formalism of *at least* taxonomic expressive power, you cannot claim to have done *domain* modeling.

  It is instructive to trace through the reasoning behind this claim. Let us assume that we have done descriptive modeling and wound up with a single concept model that has no taxonomic information in it. As a simple example, suppose we have a model that includes the concept Chair, Leg, and the relationship that chairs have four legs. We also have an exemplar set of twelve beautiful chairs from different antique periods. Have we done our job as domain modelers?

[2.] It would qualify as a "feature model," however; the facets effectively define features that differentiate the set of exemplars being classified. But if the faceted scheme is the whole representation then the "concept model" is missing!

The model in question *does* qualify as a concept model, because there is a mapping from model elements to entities in the "modeled world." That is, the concept "chair" is of the same ontological sort as an actual chair in the exemplar set, and so with chair legs, etc.

Suppose all our exemplar chairs have four legs. Then our model is in effect part of our DOMAIN DEFINITION and we have not yet done our job as domain modelers. This would be true even if the model became more and more complex and detailed, as long as all the structural information were true of *all* the exemplars. We may have received a schematic diagram from a chair-builder that names three dozen structural components of a chair, but we do not yet have a domain model.

Suppose some of our chairs are three-legged stools, and one is a rocking chair on two rails. Now our model is incomplete, because the description does not hold for all our exemplars. We have to enrich the description *or* narrow our exemplar set to make things consistent. If we narrow the set we are back in the "definition only" problem alluded to above. We therefore need to enrich the description.

There are at least two ways to do this. One is to introduce categories for "four-legged chair," "three-legged chair," and "two-legged chair." These categories form the kernel of a taxonomy (since they are all chairs). So we have created our taxonomic model. (Note here that the domain model "cheated" and called the rocker's rails "legs," clearly stretching a point. This is an example of additional "exemplar data" that need not be in the domain model, since the rocker is already distinguished adequately from other exemplars. If there were a rocker and a two-legged milking stool in the exemplar set, the distinction between leg and rail would become more critical.)

Alternatively, we could declare "number of legs" a property or feature of the chair model. The related model that "fills" this property is the mathematician's friend, "Positive Integers."

Interface

TBD

Guidelines

* A taxonomic modeling formalism that facilitates integration of specialization and relationship links in a single model will facilitate the integration. RLF and most object-based modeling systems are examples of formalisms that satisfy this criterion.

8.4 Market Analysis

Markets are arenas in which individuals and organizations develop economic relationships with each other based upon the developing and deploying of products and services. Market analysis is the segmentation of markets, using a matrix of factors and based upon assumptions and predictions about the size and change of the profit potential available within a given market segment. These market segments help businesses define their product/service sets and establish goals.

Technology markets are complex in that the products are objects, training, books, services, and even abstractions like software which conventional product or service descriptions do not adequately describe. Analyzing these markets may require techniques beyond traditional kinds of market segmentation by price, volume, geography, product, etc. Additional factors like market saturation, competition among product offerings and pricings, and substitute products are also central to analyzing markets. In technology, such concepts as leaders, early adopters, and laggards are important market analysis concepts for targeting like groupings of customer needs within differing time windows.

Approaches to market analysis include the following examples:

- Geoffrey Moore's "Crossing the Chasm" approach is particularly useful for technology-intensive offerings because it analyzes markets in terms of receptivity to innovation investment [Moor91a].

- *Scenario analysis* is a structured and creative way to ask the question "what if" and thereby be more ready for future challenges. This is particularly useful for predicting future markets and/or making necessary investments to create markets [Schw91].

- *Mass customization* is an approach to product planning and product redesign that specifically addresses some aspects of variability possible in software-intensive application areas [Pine93].

- *Value chain models* are concepts and pictures for showing relationships of different stakeholders within an increasing change of value between producer and consumer. They are pertinent to market analysis because market expansion often means moving further up or down the value chain relative to your current customers.

- The role of a domain-specific focus in the business model of an organization is also closely related to work on organizational core competencies [Prah89,Prah90].

- The work of James Withey at the Software Engineering Institute (SEI) promises to be a major contribution in the area of market analysis specifically oriented towards reuse-based software engineering opportunities. The work provides methods for evaluating and assessing economies of scope as a basis for selecting appropriate ranges of variability to support in reusable software assets. (As of the release of this document, this work was still in progress and unpublished.)

Role in ODM

Some researchers have equated domain analysis to a form of market analysis, specialized to software products targeted specifically to application developers rather than end-users of applications. The CFRP offers one specific model for reuse value chains that can be useful in market analysis for domain engineering projects [CFRP93b].

371

There are several challenges involved in applying conventional market analysis techniques to domain engineering. These include the following issues:

- Handling the greater possibilities for variability introduced by software;

- The complex nature of the software development value chain; and

- The possibility of encapsulating reusable processes as well as products.

In the ODM context, the specific market analysis aspects of domain engineering are most directly employed in the *Scope Asset Base* sub-phase, and also, in a less direct sense, in the *Scope Domain* sub-phase.

Interface

TBD

Guidelines

In light of the issues discussed above, we offer the following guidelines and caveats with regard to market analysis supporting methods:

- Recognize the market analysis task. Simple increased awareness of the need to consider explicit customers in asset base engineering may be the most significant pay-off, almost independently of the specific methods chosen. It is also important to keep several customers in mind, however, so that the real trade-offs involved in engineering for reuse remain visible.

- Do not rely exclusively on any one method. In particular, adapt the methods to the specific needs of reuse-based software engineering.

- Do not expect robustness from hard quantitative methods when applied to estimating usage of software components. Supplement these techniques where possible with qualitative data derived from other supporting methods. Stay in dialogue with the stakeholders to ensure that the project is aligned with their needs.

- Recognize when whole products need to be delivered. Throughout this guidebook there has been a strong emphasis on separation of concerns: clarifying what belongs in the province of domain engineering methods and what belongs to supporting methods, etc. In the domain engineering process itself, there is a similar emphasis on boundary-setting and scoping throughout. This separation of concerns is essential to make domain engineering a tractable engineering problem.

9.0 Optional Layers

What is a Layer?

In ODM, the notion of layers is used primarily as a way of making a separation between core and optional aspects of the method. There are two main aspects to layers that differentiate them from supporting methods:

- Layers appear as tailoring choices that have *pervasive* impact throughout the life cycle. That is, they affect aspects of how every process is performed and each workproduct is produced. Supporting methods, in contrast, seem to play a key role in particular tasks of the life cycle.

- Layers clearly extend into *optional* areas for domain engineering projects. Most supporting methods involve activities that must be performed, in some way, in order to accomplish domain engineering objectives. There are a range of techniques to choose from, and ODM is reasonably tailorable to different techniques. Optional layers will not be characterized in terms of different methods and techniques available. It is also not essential that they be incorporated to any particular extent.

This section can be used as a checklist of issues to consider in planning a domain engineering project. Within most layers various options will be presented for projects that may want to extend farther than the minimal recommendations in the core process model. Each project team will need to consider the trade-offs of resources and benefits relevant to their initiative.

Five key layers identified to date are:

- Validation

- Process and Rationale Capture

- Team Modeling

- Learning and Evolution

- Traceability

Of these, the first four (all but traceability) are discussed in significant detail in the following subsections. These subsections are organized in the form of an initial overview ("description") of the layer, followed by guidelines for applying it.

The first layer discussed, Validation, is already incorporated into the core process model to some degree in the form or task activities or guidelines addressing validation and verification issues. The sub-section here serves in part to summarize general strategies to validation provided as task-specific recommendations earlier in the document. In addition, it extends the notion of validation to optional approaches that, in effect, raise the standards by which the quality of domain models and asset bases would be evaluated.

The other layers discussed are referenced lightly or not at all in earlier sections of the document.

Background

The area of optional layers for ODM is still being defined. It is a response to lessons learned with earlier versions of the method, where it became clear that attempting to do domain engineering in

373

a systematic way can impose a considerable burden on projects, particularly when support technology for domain engineering is just beginning to emerge. Gradually, it became clear to those working on development of ODM that there was a certain amount of hidden context in the method itself, i.e., preferences and agendas that were in close alignment with, but not absolutely essential to, the primary objectives in domain engineering.

One example of such a preference is the linkage of domain engineering to the concept of the "learning organization" popularized by the work of Peter Senge and Innovation Associates [Seng90, Seng94]. The potential for synergy between domain engineering methods and this approach to learning and change in organizations is compelling. However, there will be many organizations willing to try a small-scale domain engineering pilot project that are not prepared to tackle the transition to being a learning company in the process. Accordingly, we have attempted to excise portions of the method that seem relevant *if* performing domain engineering in companies oriented towards a learning organization approach. These are included within the optional Learning layer.

Caveats

The core process model presented in the main sections of this document was significantly re-architected in version 1.0 of this guidebook to reflect this layering approach. Similar re-structuring will be required in the area of the optional layers presented here for project planners to be able to systematically apply them in various "separately selectable" combinations. In this document the layers presented are provisional only. In particular:

- Each layer represents values or concerns reflected to some extent in the core ODM process model itself. Elements included in the core are those deemed critical for achieving the stated objectives of the ODM method of domain engineering.

- There will be significant overlap between layers. For example, traceability and validation activities are interconnected in a number of ways.

- The sequence of presentation does not imply that earlier layers must be selected before later layers, or make any other implications about ways in which these optional process areas can be combined.

9.1 Validation

Description

Internal Validation

Individual models can be validated according to certain internal principles, including the following:

- *Completeness*: does the model cover all required variability as observed in the exemplar set?

- *Validity*: is every concept in the model justified by some precedent within the exemplar set?

- *Consistency*: is the model structurally complete? Are all model inter-relationships valid?

- *Expressiveness*: is the model capable of expressing variability that would be considered significant by a domain expert/informant?

- *Model balance*: is the model structurally balanced? Are there comparable and consistent levels of detail throughout the structure?

- *Minimality (Parsimony)*: Is there economical use of basic concepts in the model? Are the number of concepts constrained enough to admit a relatively intuitive grasp?

- *Intuitiveness*: do model concepts seem natural, or forced/contrived? How good is internalization of the model without recourse to documentation?

In addition, separately developed models or other workproducts are validated through integration. Integration activities can be interleaved with model development at various levels of granularity, ranging from activities performed by individual modelers to integration through project-wide review meetings. As technology for domain modeling matures, many integration tasks should become better supported by automated tools.

Process Validation

Process validation is more likely to impact the modeling process than the models *per se*. It is closely tied to the Process and Rationale Capture layer. Some useful questions to ask include the following:

- Was the process of creating the model a valid instantiation of the planned process?

- Was the actual process documented?

- Did all joint modelers achieve consensus on the model?

Stakeholder Validation

- *Formal validation of individual models*: Independent of the accuracy of the data in the models, validate them for consistency, well-formedness, style, etc. Automated tools can be useful as aids.

- Validation through integration: This is a large part of the role of the *Integrate Descriptive Models* task.

- _Empirical Validation_: Validate models with respect to the data used to create the models. Solicit feedback from informants on the veracity of the data. Check linkage from data to models.

Extended Forms of Validation

The validation activities listed above can be thought of as basic "good housekeeping" forms of validation. Extended forms of validation are possible. For example:

- Validate with exemplars that were not representatives. Use this validation to determine the robustness of the models, given unexamined data? The significant factor in this validation step is that all exemplars were presumably at least looked at in deriving the domain definition.

- Validate against new exemplars. These are systems of interest that were not considered in the original EXEMPLAR SYSTEMS SELECTION.

These subsequent levels represent a transition from what has been termed *variability engineering* to true *domain engineering*.

Guidelines

- Do internal validation before getting external feedback. Refine and validate domain engineering workproducts using internal criteria and the resources of the team before drawing in external domain experts for further validation. However, it is also wise to seek external validation before products look too finished, so that stakeholders who participate in the validation process can see that they have a voice and some ownership in the work. Modelers may get too attached to their models after spending greater degrees of effort in polishing them; and experts may be more reluctant to critique models that have the look of finished products. Factors such as the degree to which the domain expert is part of the team, schedule and resource constraints also come into play. Prepare the material as thoroughly as possible, but leave the presentation somewhat informal and open-ended, so that real feedback can be obtained.

- Use ongoing reviewers. Use some key informants who are also potential customers for periodic review throughout the modeling life cycle. This continuity of involvement helps gain buy-in and ownership from the reviewers and addresses technology transition issues while providing the necessary validation with a minimum learning curve for the stakeholders providing the validation.

- Broaden the review audience. In addition to these ongoing validation roles, use successively broader review audiences as domain engineering workproducts become more stable. In particular, try models out on some members of a potential utilizer group not previously tapped as informants. Use combined expertise in group validation and review sessions (e.g., experts not accustomed to speak with each other.

9.2 Process/Rationale Capture

Description

A certain level of skill in process capture is necessary to achieve the basic results of domain engineering. By studying processes in the development setting, modelers are able to discover opportunities for encapsulating reusable processes as well as products. Even if not specifically designed for reuse, a software artifact can be reused more predictably with documentation of the rationale behind its design.

There is a secondary layer to this emphasis on process and rationale capture in ODM, which focuses on capturing process data and rationale for the domain engineering project itself. The primary motivation for this process and rationale documentation is to preserve information that will facilitate iteration or backtracking in the life cycle when it is required. Iteration is a natural part of the process; it can reflect breakdowns or errors, but can also be a response to changes in availability of resources or even new opportunities created by successful phases in domain engineering.

This form of process and rationale capture in ODM is evidenced in a number of the core workproducts and processes in ODM discussed in the main sections of this document, including the following:

- separate documentation of candidate lists and selection criteria before selection tasks

- separation of domain-specific from resource-based prioritization steps

Many of the workproducts described as "interim" in this document thus serve a dual purpose. The primary purpose is to break the modeling process down into relatively small and well-defined activities. In addition, they provide checkpoints that capture the decision process at particular well-defined phases; each of these phases can become a roll-back point if certain project objectives or constraints shift or an impasse is reached. Keeping the various workproducts consistent does impose some hefty overhead which should eventually be addressed by better automated support.

A second level of process capture, which is an optional extension to core ODM, is oriented toward the needs of future domain engineering projects within the same organization context. The intent is to facilitate reuse of various domain workproducts in subsequent projects. For example, if a new domain is selected in a business area where the ODM planning process was carried out thoroughly, a number of workproducts will have been created that will allow the selection process to proceed much more rapidly, including the DOMAINS OF INTEREST, the DOMAIN SELECTION CRITERIA, and portions of the DOMAIN DEFINITION. With each subsequent domain modeling effort in a given business area, this reusable base of domain engineering resources should be expanded and refined.

One significant kind of reusable resource that will help improve the productivity of subsequent projects are *starter models*: abstractions from domain-specific models that provide a useful starting point for models in new domains. These might include any of the following:

- A set of optional domain-specific concept or feature models. For example, a model of the Ada type hierarchy could be developed that could be used in structuring asset bases of unit test plans for Ada components. This model would not need to be re-built for subsequent projects.

- A set of templates, intended for adaptation and tailoring; or guidelines, checklists, generative

377

tools, validation and consistency-checking tools, etc.

Selection and tailoring processes occur throughout the process model, with a recurring set of sub-processes: reusing the available resources, defining criteria for candidates, generating a candidate set, establishing characterization criteria, characterizing, ranking and prioritizing the candidates, documenting the selection rationale.

A final level of process capture, also an optional extension to core ODM, would take this documentation one step further. In this layer, an attempt is made to document changes made to the various workproducts over time. The purpose of capturing this information is to rapidly improve the domain engineering process and methods through project experience and lessons learned.

A good example of a process capture technique of this sort is a "Domain Boundary Decision Log", which documents the history of iterative decisions that shift the domain boundary over the lifetime of the project. Study of this log could reveal certain problems in the process, such as boundaries that oscillate back and forth over time and fail to come to closure.

This kind of process and rationale capture could allow for examination of the methods to improve the process. This information can be of value to individual modelers who want to improve their skills, to an organization attempting to tailor a domain engineering method suitable for their specific needs, and to the developers of the methodology.

Guidelines

9.3 Team Modeling

Description

Many aspects of the ODM method require different sorts of team interaction skills than those customary for groups of software developers. The modeling process works by continual interaction between centralized and distributed modeling tasks. Several groups may build related models relatively independently, then meet to integrate the models and resolve the inconsistencies. Model development is partitioned into reasonably separate tasks, which can be performed iteratively, in parallel or even in round-robin fashion, with circulation of personnel between different modeling and data-gathering tasks. This separate modeling strategy is necessary, even when models produced are conceptually linked together and could in principle be thought of as one large model. This is considered an optional layer because a team modeling approach can be used throughout the ODM life cycle. At every point where it is used, it can offer the following benefits:

- optimization of staff resources;

- acceleration of project schedule through parallel tasking;

- independent cross-validation, by having separate teams work on closely inter-related tasks and models;

- mitigation of limitations in support technology (e.g., for dealing with large models);

- better control of the implicit "cognitive bias" by determining the sequence and diversity of data that particular modelers will examine.

However, use of a team modeling strategy will also pose some challenges for the domain engineering project. More stringent integration activities are required to ensure the consistency of separately developed models and the traceability of elements across various models. Unlike software modules, which can have an unambiguous interface and a clear separation of concerns, related models will tend to overlap and have numerous interconnections, not all of which can be anticipated in advance. This requires a high degree of constant, detailed technical interaction on the part of the modeling teams. It also requires greater attention to process management, and greater levels of collaborative and listening skills (the latter closely related to the descriptive modeling mindset alluded to in the section on Information Acquisition techniques in Section 8). Thus, the other aspect of team modeling that makes it an optional layer is the degree to which teams are willing to address these skills. A good resource for exploring these directions is provided in the learning organization field, specifically in the core discipline of team learning [Seng90, Seng94].

Guidelines

- <u>Build Diverse Teams</u>. Diversity in team structure can enrich the quality of domain models by providing validation from multiple perspectives; however, it may also complicate the management task.

- <u>Level of Domain Knowledge</u>. Both experts and novices with respect to knowledge of the domain of focus can add value in domain modeling. Certain ODM processes are better suited for the "naive", others for the "expert" domain modeler. One might think a naive modeler must do all the work of the expert modeler, plus more to come up to speed in the domain. For certain tasks, however, modelers' prior domain knowledge of the domain can create potential perceptual biases. Teams designed to maximize diversity in this respect can produce highly robust results, by including personnel with a range of familiarity with the domain, as well as

those experienced in related or analogous domains who bring still other perspectives to the domain modeling effort.

- <u>Peripheral Experts</u>. There may be significant culture clash between applications developers and domain modelers. It may be appropriate to identify a set of expert reviewers "of first resort" for the project, welcoming but not depending on full-scale participation. For example, such a "peripheral expert team member" could be assigned the task of creating and validating the Domain Lexicon which requires extensive domain knowledge but not familiarity with the domain modeling representation formalism.

- <u>Circulation of Personnel</u>. Circulation/rotation of personnel between domain and application engineering teams, particularly for projects of some duration, will strengthen connections between the teams and a sense of buy-in by the applications groups. However, the learning curve on domain engineering projects can be particularly steep; so rotation that is too rapid can be counter-productive.

9.4 Learning and Evolution

Description

Learning is fundamental to reuse. An approach to domain engineering that does not incorporate direct support for ongoing learning about the domain runs the risk of achieving only a very limited and static kind of reuse, based on a single snapshot of knowledge about the domain. In fact, domain knowledge is constantly evolving for individual practitioners, the organization, and the technical area as a whole. Domain models and asset bases can play an integral role in accelerating that learning process, but only if infrastructure is in place to reflect the learning in the evolving asset base itself.

As each asset is used on successive applications, a base of experience grows around that asset. This might be reflected in requested changes or enhancements from asset utilizers, or in the development of ancillary material (example usage, tips on adaptation, test suites, etc.) developed and shared by utilizers. A well-designed asset base infrastructure will support the incremental addition of this supporting material over time.

Domain Evolution

Domain models will evolve over time, in addition to the assets created and managed with these models. Changes can take place with respect to several different timelines:

- over the course of the initial domain modeling process;

- over the lifetime of the assets and the asset base;

- with the evolving level of knowledge about the domain within the organization and the technical community.

Evolution of domain models and asset bases may take the following forms:

- incremental migration from informal model semantics (e.g., a checklist, outline, or keyword list) to more formal model semantics (a faceted classification scheme, or inheritance-based model);

- transformations in the classification scheme;

- gradual inclusion of more heuristic knowledge to guide modeling choices;

- changes in implementation, e.g., changing a component-based approach to a generative approach;

- generalization: promoting domain-specific models or components to the level of infrastructure, or more horizontal usage. For example, a model of error conditions developed for one real-time domain may turn out to be largely transferable to other domains with similar attributes;

- aggregate components: in an asset base that begins by implementing a base set of components, a set of "widgets" that configure multiple components can evolve over time;

- consolidation, diversification, model reorganization, and other transformations at the same implementation level.

381

<u>Learning</u>

There are several "learning loops" involved in the domain engineering and broader reuse-base development scenario:

- Individual practitioners' learning about:

  — the specific domain (i.e., the individual becomes more knowledgeable about the domain);

  — project-specific domain engineering infrastructure (the individual becomes more familiar with the infrastructure available);

  — general domain engineering methods and tools (the individual becomes a more skillful domain engineer, who could transfer this skill to other organizations and domains).

- Organization learning about:

  — the specific domain (i.e., industry knowledge about the domain advances);

  — project-specific domain engineering infrastructure (i.e., the infrastructure evolves with use);

  — general domain engineering methods and tools (the maturity level of the organization advances).

- Methodologists' learning contributing to:

  — Individual analysts' skills evolving towards transferable expertise in teaching and practicing domain engineering as a discipline; and

  — Advancement of methods and tools (i.e., domain engineering methods and tools evolve through learning feedback from practitioners).

Guidelines

10.0 Guidelines for Applying ODM

10.1 Project Planning

10.1.1 Allocation of Resources

The symmetry and balance of the ODM process tree may give the mistaken impression that the level of effort expended on planning, modeling, and asset base engineering respectively should be about equal. It would be more appropriate to say that the granularity of the activities increases the farther into the life cycle we get. Thus, though the planning steps are elaborated to the same depth in the model, they should not take as long, nor should they consume as many staff resources as subsequent phases. Conversely, asset base engineering is elaborated to a lesser degree than the other phases (primarily because substantial portions of the work in that phase are relegated to supporting methods), yet it will generally consume the most resources. The proportions suggested by the process tree are better indications of the specific content that ODM (in contrast to supporting methods) offers for each activity.

10.1.2 Scheduling and Effort Estimation

The *Plan Domain* and *Model Domain* phases together should take no more than approximately one third of the time allocated for domain engineering as a whole. Within this time, the initial planning phase should take roughly one third of the time allotted (i.e., one-ninth of the total project period of performance). We say "initial" here because a certain amount of iteration back to planning activities will recur throughout the project; this is in addition to ongoing project management activities. These proportions will not hold, of course, in situations where the project is only intended to perform steps through domain modeling.

Within the planning phase itself, expect each succeeding phase to take longer; i.e., *Define Domain* should be allocated the most time. *Select Domain* next, and *Set Project Objectives* should be the shortest process of the entire life cycle. First, but not least: even if the objectives and stakeholders are clarified in a few meetings at the start of the project, it is vital that these meetings unearth certain dynamics in the project context that will otherwise almost certainly create enormous problems farther downstream. The nature of these problems are distinct enough from those encountered on any software project that we believe the level of detail here is warranted.

The planning phase, while shorter in duration and leaner in allocated staff, requires the highest level of skill in negotiating the organizational aspects of domain engineering. As with the initial contracting in a consulting situation, mistakes in objective-setting and expectation management at the outset can stymie the entire effort. In addition, domain selection, one of the critical leverage points of the entire process, is embedded within the other planning activities. This creates a project management dilemma. On the one hand, planning activities need to happen fairly quickly and with limited staff. Yet the organizational complexities are greatest at this point, and there is a need for someone with seasoned negotiation and modeling skills to be involved. This suggests a need for access to consulting or non-line management staff resources for the initial planning phase.

10.1.3 Infrastructure Planning

TBD

10.2 Tailoring

The ODM process model can be thought of as a framework that will be tailored for each domain engineering project. Because of the inherent variability in the domain engineering process in different organization settings, this tailoring is an anticipated part of applying ODM.

Tailoring Strategies

Tailoring can take three major forms:

- A *specialization* of the ODM process model applies tailoring techniques to produce a process model that is itself a framework, but one oriented towards a narrower organization context than the general ODM method. Most typically, this will be a process model developed for the needs of a particular organization. Examples are the domain engineering workbooks/guidebooks produced within Hewlett-Packard and the Army CECOM/SED organization [Coll91a, ADER96a, DEGB95a]. Specializations could also be produced for ODM methods tailored to particular kinds of domains, or assuming particular choices of supporting methods and layers.

- An *instantiation* of the ODM process model applies the tailoring techniques to produce a process plan for a single specific project. This will be appropriate for projects that are attempting to do domain engineering using a process-driven approach. Process-driven software engineering is another facet of the STARS product line paradigm, synergistic with but independent of the focus on architecture-centric, domain-specific reuse that ODM directly supports.

- The ODM process model can also be used to document *process history* for a project. It can be applied both to projects formally conducted as domain engineering efforts, as well as to projects that performed domain engineering in an informal way, or even projects not explicitly recognized as domain engineering at the time of performance, but satisfying the bounding criteria for domain engineering established in Section 3. An early version of ODM was used in this review capacity within Hewlett-Packard, as part of the RADAR (Reusability Analysis/Domain Analysis Review) process. Process histories can be used for process improvement within the organization, comparison of different projects, or for evolution, validation and refinement of the ODM method itself.

By combining these three modes of tailoring into an integrated cycle, the combined benefits of a process-oriented approach and a systematic learning approach can be obtained. A specialized tailoring of ODM is used as the basis for deriving a domain engineering project plan. The project is performed with some resources devoted to maintaining a process history (i.e., what was done as opposed to what was planned), metrics, and lessons learned. The results are used to refine the specialized process model for the organization, to improve subsequent project planning. At the same time, the lessons learned provide feedback to refine the general ODM methodology.

Tailoring Transformations

Whether done to produce a specialization, an instantiation, or a process history, tailoring of the ODM process model can involve a number of specific transformations. The most important of these tailoring strategies have been formalized in the structure of the supporting methods, discussed in Section 8, and the optional layers discussed in Section 9. In addition to supporting methods and optional layers, the following types of transformations can be applied to the core process model presented in Part II of this Guidebook:

- Deletion of particular tasks or workproducts, considered inappropriate or irrelevant for the

organization or types of domains of interest, or not feasible given project resources;

- <u>Addition</u> of other processes or workproducts to the project plan, based on opportunities for synergy with other engineering objectives for the organization;

- <u>Sequencing</u> of tasks into a specific order of performance, including the possibilities of:

  — overlapping tasks or performing them in parallel with multiple teams;

  — iterating between or cycling through series of tasks; or

  — performing tasks retrospectively, working backward through the ODM life cycle from workproducts created following other methods or via an informal process.

  The latter is analogous to reverse engineering, but applied to domain engineering rather than system engineering workproducts.

- <u>Renaming</u> of processes or workproducts to better suit the background and culture of organizations attempting domain engineering projects.

- <u>Restructuring</u> the process model by splitting particular tasks into greater levels of detail, consolidating task descriptions, etc. The activity descriptions in the document are intended to be especially flexible; some activities have a transitional nature and could be re-allocated. As long as the underlying dynamics of the process are respected, the model should be fairly robust and flexible.

A complete guide to this tailoring process is beyond the scope of this document. In the following paragraphs, we offer a few snapshots of specific tailoring steps that could be made based on particular organizational scenarios. These descriptions presume familiarity with the ODM process model described in Part II of this document, and are for the purposes of illustration only.

Example: Transition from Domain Modeling to Asset Base Engineering

Once the *Model Domain* phase has produced a DOMAIN MODEL, there are a number of alternative scenarios for validating the model and for completing *Domain Engineering*; these depend to a large extent on the PROJECT OBJECTIVES. The DOMAIN MODEL itself will be sufficient to address some of these objectives; others will require moving on to *Engineer Asset Base*. Each of the following scenarios can be treated as a possible strategy for validation and verification, or as a desired end use of the results of domain engineering.

1) *Use the domain model directly as an asset.* The domain model can be used as a basis for maintaining legacy systems in the domain, educating new domain practitioners, etc.

2) *Build applications directly from the domain model.* Even if no assets are developed specifically for reuse as a consequence of the domain model, the application development process can benefit from having a consistent language for describing ranges of system capabilities. (Researchers who speak of domain analysis as a step in the application development life cycle that precedes systems analysis may be thinking along these lines).

3) *Use the domain model as an index back to artifacts.* When descriptive modeling has been thorough enough, the domain model can serve as a useful index back into a collection of legacy artifacts. This can support an application engineer working from a domain model who wants to utilize legacy artifacts in building desired applications. ODM, when augmented with the *traceability* layer, supports this use of the domain model.

4) *Build assets directly from the domain model.* An asset implementor can use the domain model to specify the features of the assets being developed, without the benefit of an asset base architecture. The implementor could also use traceability information, if present, to identify legacy artifacts that can serve as useful prototypes for the desired assets.

5) *Build a system architecture from the domain model.* After producing the domain model, application developers can create new systems in the domain by deriving individual system architectures from information in the domain model. Each architecture created in this way addresses the requirements of only a single application context.

6) *Build a generic architecture from the domain model.* Following domain modeling, a single "generic architecture" or "domain architecture"[1] can be developed. Such architectures utilize architecture representation techniques similar to those used for individual system architectures, but is structured to be usable within a larger class of applications within the domain. Often the generic architecture will be accompanied with variable sets of components engineered to fill specific slots within the architecture; this is the primary means of supporting variability in such an architecture. In this scenario, the asset base will usually impose an architecture that must be adopted as a standard by all applications making use of the assets.

7) *Build a variable architecture from the domain model.* Following domain modeling, a set of architectural variants can be specified that can be used to create a ***variable architecture***, a configurable architecture framework from which a class of specific system architectures can be derived (or ***instantiated***) by asset utilizers. Like the generic architecture, a variable architecture will be accompanied with components engineered to work within the architecture; however, the mechanisms for supporting variability are broader. For example, there can be variability in the overall topology or structure of the architecture, not just in the selection of components with which to populate the architecture.

The ODM process has been designed to address the entire range of alternatives described above: a domain model is developed which can be used for a variety of purposes, including engineering an asset base using a variety of implementation strategies. Depending on how ambitious a particular project's objectives are, certain processes and workproducts in the core ODM life cycle presented in this document may be of lower priority. This is particularly but not exclusively true in the *Engineer Asset Base* phase.

Example: Exemplar System Genealogy

The core ODM process recommends that detailed historical relationships between exemplar systems in the domain be documented in the *Plan Data Acquisition* task, within the *Acquire Domain Information* sub-phase of the *Model Domain* phase. (Note: these relationships are different from the DOMAIN HISTORY documented in the *Situate Domain* task of *Plan Domain*). This information is also needed in the *Interpret Domain Model* task in the *Refine Domain Model* sub-phase of *Model Domain*.

The rationale for performing this analysis in *Plan Data Acquisition* is that this information is required in order to make a good REPRESENTATIVE SYSTEMS SELECTION. As this can be a potentially labor-intensive analysis to perform, however, it is deferred as late as possible so that modelers only survey the historical connections of systems that are strong candidates for inclusion as representatives.

---

[1] The quoted phrases above ("domain architecture", "generic architecture") indicate terms used in the approaches to reuse described, but *not* used as technical terms in ODM. For example, use of the term "domain architecture" is generally avoided in this guidebook, in part because common usage of the term conflates meanings it is important to keep distinct in the ODM context.

For an organization that has a clear set of systems of interest defined as part of the organization context (e.g., a system maintenance organization with clear responsibility for a given set of legacy systems) there may be intrinsic value in a "Systems Genealogy" that details these historical relationships among all the systems. (Note: this is *not* a defined workproduct within the current ODM process model.) Such information may be of great value even to portions of the organization that are not directly involved in the domain engineering project nor interested in any of its other results.

In such a context it may make most sense to perform some of the activities required in the *Plan Data Acquisition* task much earlier in the life cycle; in fact, perhaps before determining the DOMAIN SELECTION or even the PROJECT OBJECTIVES. This is a valid and sound tailoring of the ODM process, because the key benefits of the tasks and workproducts are retained, while risks that motivate the core process model structure are addressed through other means. In this example, the key contextual difference is that there is a valid mechanism for early scoping of the set of systems that will be the subject of historical analysis.

Example: Selecting objectives and domain in parallel

The task structure of both the *Set Project Objectives* and *Scope Domain* sub-phases represent a transition from a *descriptive* to *prescriptive* orientation within the tasks. It is possible to interleave the descriptive and prescriptive tasks of the two sub-phases: i.e., first performing the *Determine Candidate Project Stakeholders* and *Identify Candidate Project Objectives*; then performing *Characterize Domains of Interest* and *Define Selection Criteria*; then deciding on the PROJECT OBJECTIVES and DOMAIN SELECTION in parallel.

This sequence could have advantages in a stable ORGANIZATION CONTEXT, where *Set Project Objectives* focuses on stakeholders for the context, while in *Scope Domain* planners develop a domain-oriented view of the same context. The results of both initial descriptive tasks can be performed once, then reused for subsequent projects in the same context. This approach requires closer interaction between the organization and domain-oriented views, along with a greater willingness to defer closure on specific commitments and selection. A risk of this approach is expending too many resources on an unfocused survey of potential domains.

10.3 Key Challenges

This section offers a few concluding remarks about some key challenges to be considered when putting ODM into practice. In particular, three pervasive challenges are highlighted that will surface in all aspects of the project.

Handling anxiety around deferring decisions

Domain engineering could almost be described as the art of intentionally deferring decisions. The tendency of engineers to "rush to code" is familiar to project managers who have attempted to introduce more systematic software development practices via CASE technology or specific design methodologies. In domain engineering this tendency will re-emerge continually and at higher levels, i.e., the "rush to design". In fact, the pressure will be intensified since, in many ways, it is only in the *Scope Asset Base* sub-phase in the *Engineer Asset Base* phase of domain engineering that a real analogy to the systems requirements phase can be found. For much of the earlier part of the process, engineers will feel themselves in uncharted territory, and the temptation will always be to fall back on familiar activities, i.e., designing solutions.

The degree to which technology decisions can and should be deferred will vary for each organization and project. The objective in ODM is not to defer these decisions by habit or fixed rules, but where there is a strategic benefit in doing so. The challenge for each project will be distinguishing those situations where late binding of decisions is appropriate.

One clear symptom to watch for that indicates this issue is operating is when the project team finds its attention continually drawn to a decision that does not appear to fit the project's place in the domain engineering life cycle. For example, engineers try to select the domain before objectives are defined; to determine the architecture before the domain model is developed, etc. We offer the following rule of thumb as a test to apply in these cases:

- Does the decision in question actually point to a project constraint?

  If so, it is legitimate to make this decision early in the process—in fact, as early as possible. If it is a real constraint, it is external to the direct control of the project and therefore not really a decision point at all. There is no point in pretending that the set of customers for the asset base will remain indeterminate until the *Scope Asset Base* sub-phase, if in fact it is known that a certain project must be a customer.

  Document the constraint explicitly in the project constraints. Then move on. The conflict has served as a means for identifying another piece of the "hidden context" for the project. This sort of information belongs in the project constraints from the point it is identified.

- Is there particular risk or uncertainty associated with a particular decision, or a large amount of lead-time required to implement a given decision?

  If this is the case, it may be legitimate to accelerate the decision point. If possible, though, consider using an iterative prototyping strategy. Specify what aspects will be determined by the prototype effort and what will not be determined. Schedule a specific point in the process when the binding decision will be made, to reinforce the fact that the prototype effort is exploratory, or for the purposes of technology evaluation only, etc.

  For example, selection of an ASSET BASE ARCHITECTURE happens very late in the "official" ODM life cycle. In practice, the decision about what architectural approach to take will have numerous repercussions, not only for the project, but for external groups that may be attempting to coordinate their schedules with project results.

If neither of the two cases above seem to apply, consider the possibility that the anxiety and focus around the decision reflects habit, preference, or just greater interest on the part of engineers in the decision in question. If this seems to be part of the problem, then treat deferral of that decision as a specific part of the discipline of domain engineering. The right strategy will not always feel comfortable or familiar. (To be fair, it is also quite possible that the conflict between perceived priorities and the suggested process model indicates an error or context-dependency in the ODM process model itself. Please provide any feedback you can to the method developers to help us to continually refine and improve the method.)

Dealing with complexity and formality in the process

Even with the separation of the core aspects of the ODM method from supporting and optional elements, the method is complex and advocates an unusual degree of formality in both representations and processes followed. This will present a significant technology transfer challenge in most settings. A few simple guidelines may help to reduce the burden this places on those learning and applying ODM:

- <u>Adopt formality incrementally, and only as required</u>. In most cases, it is possible to iterate back to previous steps and add information, increase formality, or establish linkages that were previously left informal. This may require more work than doing it the first time; but if the motivation for the extra formality is not clear at the time it is done, the results may not be worth the effort expended anyway. Formality is a technique to use when the goal is clear.

- <u>Use support technology wherever possible</u>. Employ any and all techniques to reduce the burden of the complexity on the individual engineer. The domain engineering situation by its nature introduces complexity on many simultaneous levels; human beings are generally not good at handling this complexity unassisted. Improving support technology is one of our primary areas of research for the future. Some of the formality suggested in this guidebook will not be achievable or sustainable without this extra level of automated support. Early adopters of the method will need to be willing to do some seat-of-the-pants domain engineering.

- <u>Learn to enjoy the complexity</u>. Given the two caveats above, the final suggestion is to recognize resistance to complexity when it surfaces, and to counter it with an intentional emphasis on the challenge of learning to manage greater levels of complexity and formality when engaged in the service of broader business and technical goals. Once again, if gratuitous complexity has been introduced in the method, we welcome any feedback to simplify it and make it more accessible. But to the extent that domain engineering does add a level of complexity beyond single-system engineering, enjoy it.

Integrating diverse skills

The ODM method places particular demands on domain engineering because it requires closely integrated use of skills and disciplines often associated with different staff positions and roles. Some of the specific skill areas required have been organized in terms of the supporting methods and optional layers discussed in Sections 8 and 9. Integrating these skills is challenging because some skills cross historical boundaries between organizational divisions (e.g., engineering groups versus marketing versus management) or even cultural and professional barriers (e.g., organizational and social scientists versus technical engineers). Furthermore, different aspects and phases of the method will resonate with people of different personal styles and temperaments. While these profiles should be useful in assessing candidate personnel for domain engineering projects, it will be rare to find individuals with strong skills across all disciplines. These guidelines may help in this regard:

- <u>Use team modeling and learning strategies</u>. ODM projects are best performed by multi-disciplinary teams, working collaboratively on planning, information gathering and modeling, and asset base engineering activities. Be alert to the potential of staff from non-engineering backgrounds, such as corporate librarians, technical writers, research assistants, market researchers, and even those traditionally relegated to administrative, clerical or support positions. There are portions of many activities included in the overall domain engineering process that can be effectively delegated to people with these varied backgrounds, given proper training and inclusion in the team.

- <u>Do not skimp on training</u>. Domain engineering is more than a set of formal processes, notations, and techniques. It involves developing specific perceptual abilities (e.g., taxonomic modeling skills, perceiving analogies, clustering), increased contextual awareness, and other capacities beyond purely technical skills. No person's particular background provides a guarantee that they will make the transition to domain engineering without training, guidance and team support in the process.

- <u>Treat domain engineering as a personal discipline</u>. Teaming and training strategies aside, each individual can use the domain engineering process as an opportunity to develop per-

sonal mastery in unfamiliar areas [Seng90, Seng94, Simo91a]. We believe many of the skills required in domain engineering — separation of descriptive and prescriptive processes, contextual thinking, boundary-setting — will transfer to useful general skills and disciplines. In the end, any method or technology should be judged not only by its specific results, but also by the affect that it has on its practitioners. We hope the influence of ODM as an engineering, conceptual, and social discipline will be both beneficial and enjoyable.

Appendix A: ODM Process Model

This appendix presents the ODM life cycle modeled using the Process Tree and $IDEF_0$ notations. The full Process Tree diagram and the set of $IDEF_0$ diagrams appear on the pages that follow. The order in which the $IDEF_0$ diagrams appear is consistent with the hierarchical process decomposition structure of ODM and should be straightforward to follow.

Organizations can use the Process Tree and $IDEF_0$ model directly as a basis for modeling ODM in more detail. They can also adapt the processes and information availability to address their specific needs, or they can integrate the model (or some adaptation thereof) with existing $IDEF_0$ process models.

Process Trees were developed on the Army/STARS Demonstration Project to depict multiple levels of functional decomposition of a process in a hierarchical, graphical representation. A Process Tree decomposition appears as activities connected by arrows. Arrows decompose an activity to its subactivities. Process Trees provide a convenient way of navigating complex process decompositions. Nodes within Process Trees are used to index into the ODM $IDEF_0$ Diagrams.

$IDEF_0$ is becoming an increasingly popular process modeling notation, but not all readers of this document may be familiar with it. The following is a brief overview of the $IDEF_0$ notation:

> An $IDEF_0$ activity diagram contains one level of decomposition of a process. Boxes within a diagram show the subactivities of the parent activity named by the diagram. Arrows between the boxes show the availability of information to activities. Arrows entering the left side of a box are inputs to an activity, arrows exiting the right side of a box are outputs from the activity, arrows entering the top of a box are controls that regulate the activity, and arrows entering the bottom of a box are mechanisms that support the activity. A sequential ordering of boxes in a diagram does not imply a sequential flow of control between the activities. Any activity can be further decomposed into another $IDEF_0$ diagram describing its subactivities.

See [IDEF81, Marc88] for more details on the notation syntax and semantics.

CONTEXT:

Top

Project constraints

Organization context

Organization information

Exemplar system artifacts

Domain Engineering

0

Domain definition

Domain model

Asset base

NODE: -0　　　TITLE:　　　NUMBER:

CONTEXT:

Organization context → Plan domain 1

Organization information →

Domain definition → Domain definition

Project objectives

Domain stakeholder model

Project constraints

Stakeholder dossier

Domain stakeholder knowledge

Model domain 2 → Domain model

Domain model

Domain dossier

Exemplar system artifacts →

Engineer asset base 3 → Asset base

NODE: 0   TITLE: Domain Engineering   NUMBER:

394

| NODE: | 1 | TITLE: | Plan domain | NUMBER: |

CONTEXT:

Organization context

Stakeholder dossier

Organization information

Reuse attributes

Characterize domains of interest
1.2.1

Domains of interest

Project objectives

Define selection criteria
1.2.2

Domain selection criteria

Project constraints

Domains of interest

Select domain of focus
1.2.3

Domain selection

Candidate exemplar systems

Domain stakeholder model

Project objectives

Project stakeholder model

| NODE: | 1.2 | TITLE: | Scope domain | NUMBER: |

CONTEXT:

| NODE: | 1.3 | TITLE: | Define domain | NUMBER: |

CONTEXT:

Project constraints

Domain dossier

Domain model

Domain definition

Project objectives

Features of interest

Descriptive models

Domain dossier

Domain terms

Acquire domain information
2.1

Describe domain
2.2

Refine domain model
2.3

Domain stakeholder model

Stakeholder dossier

Exemplar system artifacts

Domain stakeholder knowledge

NODE: 2   TITLE: Model domain   NUMBER:

CONTEXT:

Project objectives

Project constraints

Domain stakeholder model

Stakeholder dossier

Domain stakeholder knowledge

Exemplar system artifacts

Domain artifacts

Domain definition

Representative systems selection

Data acquisition plan

Plan data acquisition

2.1.1

Domain informant knowledge

Representative system artifacts

Elicit data

2.1.2

Domain data

Integrate data

2.1.3

Domain dossier

New system artifacts

Domain terms

Features of interest

Domain dossier

NODE: 2.1   TITLE: Acquire domain information   NUMBER:

CONTEXT:

Asset base

Project constraints

Domain interactions

Domain definition

Asset base architecture

Implement asset base

3.3

Assets

Asset base infrastructure

Architect asset base

3.2

Asset specifications

Exemplar systems selection

Assets of interest

Scope asset base

3.1

Asset base model

Asset base dossier

Asset base customer information

Project objectives

Domain stakeholder model

Domain dossier

Organization information

Domain model

Exemplar system artifacts

NODE: 3   TITLE: Engineer asset base   NUMBER:

CONTEXT:

Project constraints

Project objectives

Exemplar systems selection

Asset base dossier

Asset base model

Assets of interest

Correlate features and customers 3.1.1

Customer dossier

Prioritize features and customers 3.1.2

Select features and customers 3.1.3

Candidate customer information

Domain stakeholder model

Domain dossier

Organization information

Domain model

NODE: 3.1

TITLE: Scope asset base

NUMBER:

CONTEXT:

Assets of interest

Asset base architecture

Asset specifications

Define asset base architecture
3.2.3

Determine internal architecture constraints 3.2.2

Internal architecture constraints

External architecture constraints

Project constraint

Determine external architecture constraints
3.2.1

Asset base customer information

Domain interactions

Asset base dossier

Asset base model

NODE: 3.2   TITLE: Architect asset base   NUMBER:

CONTEXT:

Project constraints

Plan asset base implementation 3.3.1

Asset base dossier

Asset base architecture

Asset specifications

Infrastructure implementation plan

Test and validation plan

Asset implementation plan

Asset base model

Implement assets 3.3.2

Exemplar system artifacts

Assets

Implement infrastructure 3.3.3

Domain model

Asset base infrastructure

| NODE: | 3.3 | TITLE: | Implement asset base | NUMBER: |

Appendix B: ODM Lexicon

Conventions

This lexicon defines terms used in the document. The typographic conventions used within this section differ from the rest of the document, primarily for the sake of readability. Three types of terms are included in the lexicon:

- <u>General descriptive terms</u> for concepts essential to ODM appear in the lexicon in **lower-case bold type**. (Within the main body of the document, these terms when first referenced in any given section appear in ***bold italic*** type. Afterwards they may appear in regular type or are occasionally repeated in ***bold italic*** type.)

  We have tried to include only terms that are used with special meaning in the ODM context. Some of these terms are used frequently in other areas of software engineering (e.g., "system") or general usage (e.g., "organization"). They are used in a specific technical sense in ODM, i.e., to serve as an umbrella for several more specific terms, or to make key distinctions that would otherwise be difficult to convey. For example, the term "asset" is used in preference to the more familiar term "component" to denote a product of domain engineering that can take the form of a component or a generator. This usage emphasizes that the implementation technology used can vary from asset to asset. If you see a familiar term in ***bold italic***, make sure you are clear about the specific meaning it has in the ODM context.

- <u>Workproduct names</u> appear in the lexicon in **BOLD SMALL CAPITALS**, with initial large caps. (Within the main body of the document, these terms are in REGULAR SMALL CAPITALS with initial large caps.) There are three types of entries for workproducts:

  — *Main* workproducts are top-level outputs listed in the Workproducts subsection of the various task descriptions in the document. They also appear as outputs on the lowest-level IDEF$_0$ diagrams that document the process model throughout the document and in Appendix A.

  — *Composite* workproducts are higher-level data flows indicated on the IDEF$_0$ diagrams that aggregate several main workproducts (or other composite workproducts).

  — *Worksheets* are suggested formats or representations that are aids to producing the main workproducts. They do not appear as separate entries in the Workproducts subsections nor on the IDEF$_0$ diagrams. They are listed in Appendix C.

- <u>Data flow</u> names correspond to net inputs to the ODM process as described in the IDEF$_0$ diagrams. They appear in the lexicon in **BOLD SMALL CAPITALS**. Composite and component data flows are also defined in the lexicon.

- <u>Process names and workproduct names</u> within the body of lexicon term definitions follow the same conventions as in the main part of the document. Phase, sub-phase, and task names are in *Lower Case Italic* with initial large caps. Workproduct names are in Small Capitals, with initial large caps. Process names are not defined in the lexicon, since they can be found in Part II.

- <u>Formal and informal terms</u>. In many cases a term like "informant" is used extensively throughout the document, when in fact the meaning of the term is "[domain] informant." In these cases, we have chosen to provide the main definition entry for the *less* formal term, since that is the most common usage in the document. Brackets are used to indicate variant phrases that might be used for the defined term, or in some cases, the fuller and more formal

phrase elided in the common usage within the document.

- <u>Non-ODM terms</u>. A few entries have been included in the lexicon for terms used in general reuse literature that do not have special meaning in the ODM context (e.g., domain analysis, generic architecture). These are included for clarity and completeness, but we are not offering suggested formal definitions for these terms.

accidental [feature]

A feature which is an observation of a shared attribute of exemplar systems considered so far, but not a required attribute that would apply to new cases considered as well. See also essential.

acronym

A word formed from the initial letter or letters of each of the successive parts or major parts of a compound term. Used as a defined relation between terms in the DOMAIN LEXICON.

ad hoc reuse

Use of a software workproduct in a setting for which it was not originally developed, without an intermediary step of reengineering the artifact into an asset.

adequate

A FEATURE MODEL is adequate with respect to the REPRESENTATIVE SYSTEMS SELECTION when there is a feature variant within the model that corresponds to the distinct characteristics of each representative system with respect to that feature category.

analogue

A relation between two or more exemplar artifacts that perform similar functions within their respective systems, and that have been structured for comparison during the *Integrate Data* task. Can also be applied to terms in the DOMAIN LEXICON that are part of the same term cluster.

analogy domain

Formally, two domains D1 and D2 are said to be analogous if they have overlapping sets of associated exemplar systems or defining features, with neither being a complete subset of the other. Informally, an analogy domain is one related to the domain of focus through historical or cognitive associations by practitioners. For example, the family tree domain is analogous to the Outlining domain.

application

A software-based computer system developed to solve a specified set of requirements for a single context of use. Software application or system is implied, that is, the systems emphasized are software-intensive systems as opposed to systems consisting of human interactions and social organizations or strictly mechanical systems such as machinery.

The term **system** is used in this document interchangeably with **application** in this document. When used, there is no intent to imply a systems (i.e., hardware, software, practitioners) versus software-only distinction, although the focus of the document is on software-intensive domains. Generally applications or systems are assumed to be developed in response to a single set of customer requirements, though these could be negotiated via a variety of business models: contract, service, product, etc. An application could be a contracted computer system to be developed for an external customer, an internal program developed for use by an organization, or a software

product. The common theme is the **single-system context**, i.e., an intended single **usage setting**. Naturally, this is an over-simplification: products, for example, rarely have such clearly demarcated target customer settings. Still, for the purposes of domain engineering, it would seem that every organization has natural boundaries around projects, programs or contracts that make the term application or its equivalent meaningful in that organization context.

What constitutes a single application will vary from organization to organization. However, an organization domain usually represents functionality that spans multiple applications as defined *in the organization's own terms*. A formal asset base would not be termed an application in the sense used here. The use of the term is generally to contrast applications with assets designed for use within multiple applications.

See also system, application context, single-system context.

application context

Synonym for customer setting. In other words, if I am an application developer, then the usage setting and the setting in which I develop the system are my "application context" for that project.

Developers build systems, or applications, in development settings. For each application built, the application context is the development setting in which it was built, combined with the usage setting(s) in which it was intended to be used. This means that different applications, built in the same development setting, can have different application contexts.

The application context is the context that needs to be recovered when we study an exemplar system in order to be able to correctly interpret the features that occur within that system and to compare it to other systems. See customer setting.

architectural style

Researchers in software architectures have recognized dramatic structural similarities in the architectures of systems built to perform diverse real-world missions with little or no historical relationships, developed and used by diverse communities. Some of these structural patterns are beginning to be codified and catalogued as architectural styles [Shaw96].

architectural variant

An architectural description that differs from another architecture description with respect to some architectural features. For example, if one exemplar system was client-server based and another was not, each could be an architectural variant with respect to the architectural feature "encapsulation strategy".

architecture

Within the context of ODM, an architecture refers to a structure for interconnecting a set of constituent components. The components can be workproducts from any point in the software life cycle. Thus, it is legitimate to speak of a **requirements architecture**, meaning a collection of requirements organized into an overall structure (e.g., a requirements document); a **test-case architecture**, denoting an arrangement of individual test cases according to some organizing principle appropriate to the testing methodology in use, etc. The conventional notion of an architecture familiar to most readers would be a **design** and/or **implementation architecture**, that is, an arrangement of software design workproducts or code modules according to an overall structuring principle.

The term architecture is usually applied to entire systems. In the ODM context, where the domain of focus may consist of a sub-system scope of functionality, the term **architecture** can be applied in a relative sense to denote a structuring principle spanning the entire sub-system in the domain

setting. This means that architectural variants produced during the *Engineer Asset Base* phase will be architectures for the domain scope, not necessarily architectures for entire systems within which the domain functionality occurs. The interface between domain functionality and the surrounding system setting will usually be addressed as part of an architectural variant.

With these exceptions, ODM's notion of an architecture is compatible with most current work in the area of software architectures. Consult [Shaw96, Clem95a, Perr92a] for a more detailed understanding of architecture issues.

architecture-centric

An approach to reuse that emphasizes the development of asset bases organized around specific domains and supporting reuse of entire system architectures and components within those architectures.

architecture feature model

A feature model differentiating exemplars on the basis of architectural features.

artifact

Broadly defined, a system workproduct, created by a domain practitioners (as opposed to a member of the domain engineering team) used as a source of information for domain engineering. Using a workproduct as an artifact for domain engineering means interpreting it in ways quite different than its original purpose within its own setting.

Artifacts are generally (but not always) legacy system workproducts. Examples of artifacts are code components, bug reports, and requirements documents. The legacy system could be a system not currently in use or a system never deployed (canceled prototype, etc.). A requirements document for a system still to be developed would also be an artifact from a domain engineering standpoint. An artifact is distinct from an **asset** in that it is not assumed to be under the control of a reuse asset manager. It is not incorporated into an asset base, and has not been specifically reengineered for reuse.

The term artifact denotes the fixing of information in actual documentary form (such as on-line textual information). Domain information (e.g., knowledge about the domain in an expert's head or expressed in a conversation) must be fixed into the form of some artifact (an interview report or transcription), before it can be dealt with concretely in domain engineering. At this point it becomes an artifact.

Artifacts include:

- **Primary artifacts** which are used as workproducts in creating exemplar systems, including tools. Primary artifacts are also the subject of direct comparative modeling during the *Describe Domain* sub-phase, and are typically candidates for reengineering into assets.

- **Secondary artifacts**, including review articles, commentary, surveys etc. Secondary artifacts can include exemplar-specific artifacts as well as artifacts associated with particular informants but not with one particular exemplar system.

Other artifacts to be considered can include **legacy models**, informal or formal domain modeling efforts by domain stakeholders (earlier domain models, architectures, etc.) These must be handled differently from other artifacts because they contain direct taxonomic modeling information. For example, a **generic architecture** developed by a design team using informal reuse techniques could not be easily compared with a representative system built for a single usage setting. Also, artifacts generated by the domain engineering team itself (e.g., via reverse engineering) should be distinguished from primary artifacts.

See also primary artifact, secondary artifact, legacy model.

asset

A workproduct developed or reengineered for reuse and placed under management within an asset base. Assets can be developed from workproducts from any phase of the software engineering life cycle, including requirements, design, code components, test cases, etc. Assets can also be reengineered versions of informal engineering artifacts such as checklists, process descriptions, and guidelines.

Assets can include both static components, reused by being adapted and incorporated into deliverable systems, and tools and generators, that are reused by application developers executing them as programs to generate tailored assets. Three types of assets are engineering assets, process assets and learning assets. See also engineering assets, process assets, learning assets.

asset base

Reuse works best when collections of assets are developed, focused in particular domains. It is difficult to get isolated components reused, because the overhead needed to locate, retrieve, and adapt these components for new applications may outweigh the savings derived from their use. If a number of related components are developed for focused classes of applications, however, developers will have stronger incentives to reuse, as they can count on a significant percentage of their application being addressed by the collection of components as a whole. The ODM life cycle assumes that multiple assets will be used in concert in particular applications. We refer to a designed collection of assets targeted to a specific domain as an *asset base*.

ASSET BASE

A composite workproduct that includes the full set of ASSETS for a domain, together with the ASSET BASE ARCHITECTURE that integrates the ASSETS and ASSET BASE INFRASTRUCTURE required for the domain. The ASSET BASE also includes the ASSET BASE MODEL and ASSET SPECIFICATIONS. The ASSET BASE is developed during the *Engineer Asset Base* phase.

asset base architect

A domain engineer who defines an ASSET BASE ARCHITECTURE.

asset base architecture

Defines the rules and constraints for composing or generating assets to support domain functionality within multiple applications

ASSET BASE ARCHITECTURE

The ASSET BASE ARCHITECTURE models the range of variability among system architectures or configurations that can be derived from feature combinations in the ASSET BASE. It includes the rules, relationships, and constraints among ASSETS that govern how they can be combined and instantiated to produce application artifacts supporting specific feature ensembles. The architecture supports separately selectable feature ensembles. It should be as implementation-neutral as possible. The ASSET BASE ARCHITECTURE is developed in the *Define Asset Base Architecture* task in the *Engineer Asset Base* phase.

asset base customer

A stakeholder organization or group within a system development effort that is an intended utilizers of assets from the asset base being developed. The asset base customers provide the specific market for the *Engineer Asset Base* phase. Also referred to as customer.

ASSET BASE CUSTOMER INFORMATION

A component data flow of CANDIDATE CUSTOMER INFORMATION, this includes new information elicited from selected customers about architectural constraints of systems targeted for implementing or migrating domain assets.

ASSET BASE DOSSIER

The ASSET BASE DOSSIER includes the following information:

- A thorough characterization of each feature set and correlated customer settings. This dossier includes information in the CUSTOMER DOSSIER produced in *Correlate Features and Customers* and extends it with additional customer and feature attributes (including information from the DOMAIN DOSSIER) and information about feature utility and feasibility within each setting.

- A mapping between the feature sets and the customer settings to which they apply, prioritized in terms of key criteria such as utility and feasibility. This is a refinement of the initial mapping between features and customers produced in the previous task, *Correlate Features and Customers*. The FEATURE/CUSTOMER MATRIX worksheet template in Exhibit C-13 suggests a format for expressing this information.

Information in the DOMAIN DOSSIER about those representative systems studied during *Model Domain* that are relevant to the candidate customer settings should be carried over into the ASSET BASE DOSSIER.

asset base engineering

Another term for the *Engineer Asset Base* phase of the Domain Engineering life cycle. The *Engineer Asset Base* phase includes scoping and architecting of the ASSET BASE, and implementation of ASSETS and the ASSET BASE INFRASTRUCTURE.

asset base engineering team

Members of the domain engineering project who are performing asset base engineering.

ASSET BASE INFRASTRUCTURE

A workproduct that includes any portion of the environments, tools, documentation, and procedures required for ongoing integrated management of the ASSET BASE that are most effectively developed in tandem with the specific ASSETS of the ASSET BASE. These may include:

- Domain-specific extensions to general infrastructure mechanisms and procedures selected.

- Technology-specific extensions to the ASSET BASE ARCHITECTURE, reflecting additional constraints and semantics imposed by technology choices on the architecture.

- Inputs to a technology transfer plan. The technology transfer plan is developed outside the domain engineering life cycle, in Asset Base Management. Asset Base Management receives and transitions the ASSET BASE to asset utilizers.

The ASSET BASE INFRASTRUCTURE is developed in the *Implement Infrastructure* task in the *Engineer Asset Base* phase.

ASSET BASE MODEL

The ASSET BASE MODEL includes the following information:

- The set of features to be supported by the ASSET BASE, as well as relationships and con-

straints among the features.

- The set of asset base customers, characterized in terms of attributes related to the settings in which the selected features will be applicable (e.g., role in life cycle phases, experience level, etc.).

- A mapping of selected features to selected customers and settings. This is a refinement of the mapping between prioritized features and customers in the ASSET BASE DOSSIER produced in the *Prioritize Features and Customers* task. The FEATURE/CUSTOMER MATRIX worksheet template in Exhibit C-13 offers a format for expressing this information.

asset delivery mechanism

Provides any needed support for delivering the needed ASSET instantiation to the customer at the point of demand. This can include documentation of suggested integration, validation, and tailoring strategies.

asset implementation

Can take the form of a software component, a parameterized template, a generative tool that produces an asset instance at the point of demand, or any of a variety of other forms. An asset implementation must satisfy the asset interface specification. There may be multiple implementations per specification to support different implementation technologies or features within different development settings.

ASSET IMPLEMENTATION PLAN

Details of the technology to be used and the implementation schedule for each ASSET. Includes information such as:

- A description of the technology to be used and a general plan for how the technology is to be applied to implement each asset.

- A schedule for asset implementation that is correlated with estimated schedules for customer applications.

- A phased asset implementation and evolution strategy that describes which assets will be implemented when and the phases they may go through, based on customer need, technology availability, feedback cycles, and so on.

The ASSET IMPLEMENTATION PLAN is created in the *Plan Asset Base Implementation* task in the *Engineer Asset Base* phase.

asset interface specification

The interface specification explicitly documents the intended scope of applicability for the ASSET, the range of application contexts for which it has been designed. It is expressed in terms of domain-specific features as documented in the DOMAIN MODEL. See also scope of applicability.

ASSET SPECIFICATIONS

ASSET SPECIFICATIONS contain specifications for ASSETS that implement the selected feature ensembles. The ASSET SPECIFICATIONS include the required range of variability in the ASSETS, and constraints and interdependencies with other ASSETS. Each of these specifications is subject to an independent technology selection process in the next sub-phase, *Implement Asset Base*. The ASSET SPECIFICATIONS are developed in the *Define Asset Base Architecture* task in the *Engineer Asset Base* phase.

asset supporting material

Provides additional forms of support to application engineers in reusing the ASSET. For example, this could include example applications or a list of potential customer sites derived from the ASSET BASE DOSSIER.

asset test and validation plan

This plan provides supplemental documentation of the contextual assumptions and semantics of operation for a single ASSET. It is derived from the overall TEST AND VALIDATION PLAN.

asset utilizer

An asset base customer who is using assets from the asset base being developed.

ASSETS

Each ASSET workproduct contains an asset interface specification, asset test and validation plan, one or more asset implementations, an asset delivery mechanism, and asset supporting material. ASSETS are developed in the *Implement Assets* task in the *Engineer Asset Base* phase. See also asset interface specification, asset test and validation plan, asset implementation, asset delivery mechanism, asset supporting material.

ASSETS OF INTEREST

The set of assets seen to be desirable end products of the *Engineer Asset Base* phase, based on the feature and customer selection. The ASSETS OF INTEREST workproduct is produced in the *Select Features and Customers* task in the *Engineer Asset Base* phase.

borderline exemplar

A system that has features that provide evidence both for membership and non-membership in the domain, but not sufficient evident for either.

BUSINESS OPPORTUNITIES TABLE

A worksheet that can be used to show the interrelationships between customer settings in the CUSTOMER DOSSIER, expressed in terms of value chains associated with candidate feature sets derived from (i.e., a subset of the features in) the DOMAIN MODEL. These value chains can be viewed as a set of candidate business scenarios or business opportunities for the asset base. A template for the BUSINESS OPPORTUNITIES TABLE worksheet is illustrated in Exhibit C-12. The BUSINESS OPPORTUNITIES TABLE worksheet can be developed during the *Correlate Features and Customers* task in the *Engineer Asset Base* phase.

candidate

Under consideration for selection. See also potential.

CANDIDATE CUSTOMER INFORMATION

A data item which contains supplemental information from possible customers about feature preferences and priorities. CANDIDATE CUSTOMER INFORMATION includes configurations of features that must be available as an ensemble and features that must be excluded from ensembles for them to be usable by customers. CANDIDATE CUSTOMER INFORMATION is a component of the ORGANIZATION INFORMATION data item.

CANDIDATE EXEMPLAR SYSTEMS

A workproduct which indicates systems of interest that intersect the domain of focus, characterized with respect to this intersection. Indicates systems of interest that intersect the domain of focus, characterized with respect to this interaction. The CANDIDATE EXEMPLAR SYSTEMS are selected in the *Select Domain of Focus* task in the *Plan Domain* phase. See also exemplar system, system of interest.

canonic term

A lexicon term within a term cluster selected as the standard to be used in normalized references in lexicon definitions for other terms within the term cluster, and also in the appropriate CONCEPT MODEL. For example, in the Outlining domain, the terms "sub-head", "sub-heading", "child", and "daughter" might be used in different systems to denote the same entity in outline documents. "Sub-head" could be selected as the canonic term, and would then be used consistently in all lexicon references. See also term, term cluster.

classification

In the core ODM life cycle, descriptive modeling involves studying a set of exemplars for the domain and modeling their commonality and variability. In its purest form, this activity is generally known as classification. Classification has been broadly defined in [Bail94] as follows:

> "In its simplest form, classification is merely defined as the ordering of entities into groups or classes on the basis of their similarity. Statistically speaking, we generally seek to minimize within-group variance, while maximizing between-group variance... [making] groups that are as distinct (non-overlapping) as possible, with all members within a group being as alike as possible."

Classification is part of the Concept Modeling supporting method. See Section 8.3.

closure

A property that a feature model has when performing innovative transformations on feature model members only yields other feature model members. Used by analogy to the mathematical notion of closure under a set of operations.

commonality

Two features have commonality if they have been observed to be similar across exemplar systems. See also variability.

community of practice

A group of practitioners in a given organization setting that share terminology, knowledge, a set of behaviors and interests in a certain domain.

component

Denotes a static ASSET within an ASSET BASE, as opposed to a generative ASSET that produces an asset instance through execution of some program. Often used in the context of component-based versus generator-based implementation strategies.

COMPONENT CONSTRAINTS TABLE

Contains key constraints pertaining to feature ensembles, such as whether they can be selected as stand-alone or separately selectable. A template for the COMPONENT CONSTRAINTS TABLE work-

sheet is illustrated in Exhibit C-16. The COMPONENT CONSTRAINTS TABLE worksheet can be used in developing the INTERNAL ARCHITECTURE CONSTRAINTS workproduct, during the *Determine Internal Architecture Constraints* task in the *Engineer Asset Base* phase.

component sub-domain

A domain related to the domain of focus by implementing a well-defined portion of the functionality in the domain. Typical sub-domains would correspond to subsystems within the larger system that implements a given domain. For example, check processing might be a sub-domain of the accounting domain.

More formally, a component subdomain implements a subset of the features of the domain of focus. It may also address a broader set of application contexts than the domain of focus. In the example above, check processing may also be a sub-domain of many other domains, such as home financial accounting software. See also specialization sub-domain.

composite workproduct

A workproduct composed of a set of workproducts that are outputs of more than one task. Used in the context of the ODM process model.

concept

A concept is an abstract idea generalized from particular instances. Concepts can only be talked about through shared definitions. We cannot "get at" concepts directly; only through social interactions and agreements, mediated by representations.

People define concepts by grouping a set of individual descriptions of phenomena (exemplars) according to some principles for determining whether or not an individual description is an exemplar (defining rules). A concept definition "embraces" an individual description if the description is an exemplar for that concept.

- The set of all possible individuals embraced by the concept definition is the concept's extent. The set of individuals considered and determined to be embraced by the concept is the "exemplar set." The exemplar set is always a finite subset of the extent.

- The minimal set of properties that provide a decision procedure for new individual descriptions to determine whether or not they are embraced by the concept is the intent. Just as we can only get at the extent through the exemplar set, the intension can only be approximated by an articulated set of defining rules.

A concept "lives" in a set of primary tensions: between the extent and the exemplar set; between the intent and the defining rules; and, most accessible and useful, the tension between the exemplar set and the defining rules (that is, the tension between the intension and extension). Working with this latter tension, in fact, is a primary way to incrementally refine both the exemplar set and the defining rules so that they better capture the intended concept to its full extent. See also extent, intent.

concept area

One of the CONCEPTS OF INTEREST selected at the start of the *Model Concepts* task. The term is used instead of "concept" specifically because these are treated more as areas of focus for clustering related concepts than as root concepts of a hierarchy for each area.

Concept areas will not necessarily correlate to names of separate models; neither is there any assumption that all concept areas will wind up in one model. Each concept area will eventually be modeled by one or more concepts, which will in turn find their way into DESCRIPTIVE MODELS, not necessarily in a one-to-one correspondence. For example, "error message" could be the con-

416

cept of interest for a model focused on classifying different types of error messages provided in a system. Alternatively, modelers could choose to embed this name within a more general taxonomy, such as "system condition". The choice of how to partition the set of models is part of the modeling task and not specified by ODM. Some concepts will turn out to be related within one model; other models will be created that are not in the list. See also concept, CONCEPTS OF INTEREST.

concept model

A model created to capture semantics of the domain. Each concept model includes one or more of the CONCEPTS OF INTEREST as well as related concepts in those areas. The structure, content representation strategy for each concept model are determined by the Concept Modeling supporting method (or formalism) selected. Different models may invoke different formalisms.

concept modeling formalism

A concept modeling formalism describes a set of element types (or "concept capturing constructs") and a set of distinct semantic relationships, or structuring relation types that can be defined between these constructs. The formalism also includes constraints on configurations of the elements and relations. A concept modeling formalism is an element of the Concept Modeling supporting method.

concept modeling representation

A concept modeling representation provides a way of documenting a specific model defined in terms of the formalism. A concept modeling representation is an element of the Concept Modeling supporting method.

CONCEPT MODELS

The specific models created to capture the semantics of the domain. Each concept model includes one or more of the CONCEPTS OF INTEREST as well as related concepts in those areas. The CONCEPT MODELS workproducts are created in the *Model Concepts* task in the *Model Domain* phase.

concept of interest

See concept area.

concept starter set

A pre-defined repertoire of conceptual categories, thought to be domain-independent and sufficiently comprehensive to capture the semantics of any given setting. In practice, we almost always begin modeling with some concept starter set; different disciplines, supporting methods and representations have their own versions of these concept starter sets. Most domain analysis methods embed particular starter sets of this kind. Because they are often closely associated with a particular modeling formalism, concept starter sets are treated as one component of the Concept Modeling supporting method area described in Section 8.3.

CONCEPTS OF INTEREST

A list of **concept areas** to be developed. The CONCEPTS OF INTEREST workproduct is produced in the *Model Concepts* task in the *Model Domain* phase, and used by modelers in allocating modeling tasks, filtering data, and model development. Concepts used for a domain engineering model may range from concepts for concrete artifacts, such as code modules or other system artifacts, to more abstract conceptual entities such as types of errors, objects, or operations in the domain. The

purpose of enumerating the concept areas to be developed are to make sure all essential topic areas are being addressed, to provide a basis for balancing the modeling effort spent, and to provide criteria for knowing when you have drifted out of range of relevant topics.

Each concept area in the list is documented with the following elements:

- *Name*: a term or short phrase that captures the intended semantics of the concept area.

- *Derivation/Rationale*: Why are we interested in this area? In particular it is useful to know which concept areas were suggested by general (domain-independent) sources and which came out of examination of domain data. We list this as two pieces of information: the derivation "rule" and an indication of the specific source involved. Note that a given rule might be applied to several concept areas and that any one concept area might be "motivated" by more than one derivation source.

- *Ontological Description*: What types of entities would serve as instances of the concept? For example, suppose one defined a model of Wines in terms of wine regions (e.g., Bordeaux); another model might use the same term as the name of a generic category of wines, where particular vintages would be the individuals. Still another model could use the term to refer to individual bottles from a single vintage (e.g., for a quality control group). In each case, although the general subject is wine, the specific semantics of what an individual in the model stands for is different.

- *Illustration of entities*: Prototypical examples/counter-examples or informal feature descriptions as a supplemental way to define the concept's ontology.

- *Informal relationships between concept areas*: We can't be certain about the relationships between these things at the starting point of concept modeling. Note suspected relationships, without trying to be formal, systematic or exhaustive.

See also concept, concept area.

CONCEPTUAL/HISTORICAL RELATIONS MATRIX

A worksheet that can be used to validate the DOMAIN CLASSIFICATION by mapping domains to their evolution in the DOMAIN HISTORY. A template of the CONCEPTUAL/HISTORICAL RELATIONS MATRIX is illustrated in Exhibit C-10. The CONCEPTUAL/HISTORICAL RELATIONS MATRIX worksheet can be used during the *Situate Domain* task in the *Plan Domain* phase.

context

The context of a work product is the development setting in which it was created, and the anticipated settings in which it was intended to be used by its creators. As an example, in a typical "contract" life cycle, a development setting directly responds to a single usage setting (a single customer environment). Then the software developed in that setting will tend to be "single-use" in scope. Suppose, in contrast, that in a product life cycle, developers create a product and then hand it over to a marketer who tries to sell it to as many usage settings as possible. This latter market can be viewed as another context, a fairly informal "set of possible settings."

This notion of grouping possible settings is critical to the transition from domain modeling to asset base engineering. The concept of settings can be useful even when we are dealing with hypothetical settings or markets defined by setting characteristics rather than individual, concrete work environments.

See also development setting, usage setting, context recovery.

context recovery

A primary purpose of domain modeling is context recovery to identify and make explicit contextual information embedded within artifacts. Software created for a single application embeds hidden "contextual" dependencies that create problems when developers attempt to transfer that software to new applications. Doing context recovery for system artifacts can render them more dependable and predictable, even for *ad hoc* reuse, by making dependencies explicit. (It does not remove these dependencies; of course. This requires the further step of reengineering artifacts into assets.)

Documentation of context is also important throughout the modeling process. The language, values, assumptions and history of each relevant community introduces contextual information into software artifacts that invisibly constrain those artifacts. This "hidden context" in software artifacts is one of the primary sources of uncertainty in *ad hoc* reuse of legacy components. Because many constraining assumptions come from the cultural context in which software artifacts are developed and used, the domain modeling process must identify and make explicit how this kind of information is embedded within the artifacts.

contextual knowledge

Knowledge about the history or rationale behind a system or artifact that is not codified in the artifact itself. Artifacts can be distinguished as low-context or high-context data. High-context data requires that the person interpreting the data understands a lot of the implicit context. Low-context data assumes less knowledge on the part of the person interpreting the data. Contextual knowledge is what has to be gathered from practitioners to turn high-context artifacts into low-context artifacts.

core customer

A stakeholder that must be satisfied as an asset base customer in order for the domain engineering project to be successful. Usually mandated in the PROJECT CHARTER or otherwise imposed by the PROJECT CONSTRAINTS, PROJECT OBJECTIVES, or, most informally, by the ORGANIZATION CONTEXT. Identifying core customers is one strategy for performing the tasks in the *Scope Asset Base* sub-phase.

core exemplar

A system that modelers believe strongly defines or characterizes the essence of the domain (e.g., an established industry lead product).

core feature

A feature of interest to a core customer. Alternatively, one of the necessary inclusive DOMAIN DEFINING RULES. See also core customer.

core ODM process model

The core ODM process model addresses processes and workproducts that specifically address domain engineering concerns. The core is represented by a process model for the domain engineering life cycle. This process model can be tailored and instantiated in a variety of sequences and project structures. Since each organization and each domain will lead to unique constraints and preferences, the core domain engineering process model can be integrated with a broad variety of supporting methods. See also domain engineering, supporting method, domain engineering life cycle.

counter-exemplar

A candidate exemplar system with features identified that provide sufficient evidence of non-membership in the domain.

customer

In the ODM context, a synonym for asset base customer. Customers are stakeholders selected as targets for the asset base to be developed.

CUSTOMER DOSSIER

This workproduct captures the following information:

- Initial characterizations of the candidate customers and the domain-relevant settings in which the customers are involved.

- Interrelationships between customer settings, expressed in terms of value chains associated with candidate feature sets derived from (i.e., a subset of the features in) the DOMAIN MODEL. These value chains can be viewed as a set of candidate business scenarios or business opportunities for the asset base. The BUSINESS OPPORTUNITIES TABLE worksheet template in Exhibit C-12 suggests a format for expressing this information.

- A correlation (or mapping) between customers/settings and candidate feature sets, reflecting the business scenarios. This mapping shows the anticipated degree of interest that the candidate customers (in their relevant settings) have for the candidate feature sets. The mapping is *many-to-many*: Each feature set may be deemed of interest to multiple customers. Conversely, each customer may be interested in multiple feature sets.

  — For feature sets that are components of the domain, the more components that interest a given customer the more likely it is that the customer will utilize multiple assets in subsystem level aggregations.

  — For feature sets that are specializations/generalizations of the domain, a customer's interest in more than one feature set indicates the degree of variability that is required in the ASSET BASE.

  The FEATURE/CUSTOMER MATRIX worksheet template in Exhibit C-13 suggests a format for expressing this correlation information.

See also component, generalization, specialization.

CUSTOMER/EXTERNAL INTERFACE MATRIX

Documents the relevant constraints each customer in the ASSET BASE MODEL has on each external interface. A template of the CUSTOMER/EXTERNAL INTERFACE MATRIX is illustrated in Exhibit C-15. The CUSTOMER/EXTERNAL INTERFACE MATRIX worksheet can be used during the development of the EXTERNAL ARCHITECTURE CONSTRAINTS workproduct, during the *Determine External Architecture Constraints* task in the *Engineer Asset Base* phase.

customer setting

Any setting within a customer system's life cycle. This does not only refer to usage settings for a customer system, since the direct customers for an asset base are typically application developers working in the development setting for that customer system. A customer setting can be any setting where there are relevant features of interest. See also development setting, feature of interest, usage setting.

DATA ACQUISITION PLAN

A plan detailing information sources to be investigated for the domain. It includes the following:

- Data acquisition goals, which state what information is required from each information source, in sufficient detail to guide data elicitation. Typical goals include discovering native terminology, determining limitations of operations and filling in steps of procedures. Goals can also contain information about the scope of the investigation, e.g., limiting the investigation to certain features of a system. Data acquisition goals are usually hierarchical (including high-level goals for the entire investigation, down to low-level goals for a particular session), and may be written in outline form. A goal should indicate which system or set of systems from the REPRESENTATIVE SYSTEMS SELECTION and which information sources from the list in the DOMAIN DOSSIER are to be investigated.

- A list of all information sources for the systems in the REPRESENTATIVE SYSTEMS SELECTION. Information sources include artifacts (including user documentation, manuals, executable code, source code, design documents, presentations, tutorial materials, etc.) and informants. Each information source should be annotated with the system setting for which it can provide information.

The DATA ACQUISITION PLAN is developed in the *Plan Data Acquisition* task in the *Model Domain* phase.

data item

Data used in the ODM process that is not produced in the ODM process.

defining rule

A statement about various combinations of features that imply membership in the domain. A defining rule can have the following attributes (where "feature" can also refer to a feature combination):

- *Inclusive*: the feature holds for exemplars

- *Exclusive*: the feature does *not* hold for exemplars

- *Necessary*: the feature holds/does not hold for *all* exemplars

- *Sufficient*: All systems for which the feature holds/does not hold are exemplars

All of these rules contribute to the intensional definition of the domain. They express inherent attributes of the domain, and not merely accidentally shared features observed in particular set of exemplars which are not relevant to the domain definition. See also intensional definition.

descriptive modeling

In order to emphasize the particular "cognitive style" necessary for domain engineering, the ODM process model separates the life cycle into two distinct phases: a *descriptive* domain modeling phase, and a *prescriptive* asset base engineering phase. In descriptive modeling, we are selecting and studying a set of example systems for the domain in order to derive the shape of the domain "space" itself. After having established this, we are then in a position to carve out particular sub-regions of this space for engineering efforts.

Descriptive modeling documents what experts have learned about how to build particular classes of applications through the experience of developing multiple systems. In descriptive modeling, knowledge about the domain is obtained in part by analyzing legacy systems through the intermediary definition of domain *features*, significant differentiating capabilities across domain systems.

Modelers document commonality and variability in system structure and function and attempt to recapture the rationale for decisions embedded in those systems.

Descriptive domain modeling is thus somewhat analogous to reverse engineering in a reengineering context; or to the "current physical" and "current logical" models (sometimes called the "as is" model) in Yourdon systems analysis techniques. However, there are significant distinctions as well. In ODM descriptive models can contain information about requirements for new and anticipated systems as well as legacy systems. Conversely, customer systems for the asset base might include legacy systems to be reengineered.

The descriptive model is best thought of as a language for expressing a *coherent space of possibilities* within the domain. This model can include aspects of *both* the problem and solution space, as well as interpretive rationale connecting them. The key principle of the descriptive phase is that no binding decisions or commitments are made about the functionality to be supported in the asset base.

Note that the final domain model is extended beyond a purely descriptive model through innovative transformations applied directly to the model. This extension step is necessary to ensure that the prescriptive asset base model is always a subset of the domain model. This step is described more fully at the end of this section.

When used informally rather than to refer to the specific *Describe Domain* sub-phase, the term *descriptive modeling* usually refers to the descriptive "mind-set" or approach to modeling, as contrasted with the prescriptive engineering approach. See also prescriptive modeling.

DESCRIPTIVE MODELS

A composite workproduct consisting of the DOMAIN LEXICON and FEATURE MODELS produced during the *Describe Domain* sub-phase of the *Model Domain* phase.

development setting

The setting in which software systems are developed. Typically the programming platform and environment.

differentiating feature

A feature that holds for only some exemplar systems. Typically, but not always, a differentiating feature can be modeled as a specialization of a defining feature for the domain.

> *Example.* In the Outlining domain, all outlining programs allow for expansion of sub-heading structure. This would be a defining feature of the domain. However, some outlining programs might include an "expand-all" operation and some might not. This specializes the defining feature; that is, no outlining program could support expand-all that did not support the more general expand operation; while the reverse is not true. Expand-all therefore is a differentiating feature for the domain.

In general, while domain defining features are modeled in the *Define Domain* sub-phase of *Plan Domain*, differentiating features are not modeled systematically until the *Describe Domain* sub-phase of *Model Domain*.

diffused domain

A type of horizontal domain relation. A domain D is diffused with respect to a system S if the functionality of D is pervasive or global within S. A domain D is diffused with respect to a set of systems Sn if the functionality of D is pervasive or global within all systems in Sn. Note that, in ODM, this term is only meaningful as a relation between a domain definition and a system or set of systems. See also horizontal domain, encapsulated domain, vertical domain.

domain

An abstraction that groups a set of software systems, or some functional areas within systems according to a domain definition shared by a community of stakeholders. The domain can be considered to include not only the shared terminology and definitions, but the coherent body of knowledge about domain systems shared by that community. A domain can be:

— represented by a domain model;

— exemplified by a set of exemplar systems;

— defined by a set of domain defining features and defining rules;

— implemented by an asset base.

See also domain of interest, domain of focus, organization domain.

domain analysis (non-ODM term)

The process of identifying, collecting, organizing, analyzing, and modeling domain information by studying and characterizing existing systems, underlying theory, domain expertise, emerging technology, and development histories within a domain of interest. A primary goal is to produce domain models to support the development and evolution of domain assets.

NOTE: Since this term is used in ambiguous ways in the domain engineering community the term is avoided in this document. Instead, the ODM domain engineering life cycle includes *Plan Domain*, *Model Domain* and *Engineer Asset Base* phases.

domain architecture/domain architecture model (non-ODM term)

Since these terms are used in ambiguous ways in the domain engineering community the terms are avoided in this document. See instead: exemplar system architecture, asset base architecture. The closest ODM equivalent to the most common interpretation of the term "domain architecture" would be a "prescriptive architectural feature model."

DOMAIN ARTIFACTS

A data item that contains sources of information about the domain not associated with a specific representative system; i.e., not a system workproduct. See also artifacts.

DOMAIN CLASSIFICATION

This workproduct situates the domain of focus with respect to the *conceptual* relations outlined above: i.e., the specialization, generalization, and analogy domains. This workproduct can basically be a simple list of domains conceptually related to the domain of focus. The DOMAIN CLASSIFICATION workproduct is produced during the *Situate Domain* task of the *Plan Domain* phase.

Exhibit C-9, RELATED DOMAINS/DEFINING RULES MATRIX, contains a worksheet template that can be used to characterize and classify related domains in terms of the DOMAIN DEFINING RULES. Exhibit C-10, CONCEPTUAL/HISTORICAL RELATIONS MATRIX contains a worksheet template that can be used to validate the DOMAIN CLASSIFICATION by mapping domains to their evolution.

DOMAIN DATA

A workproduct produced during the *Elicit Data* task of the *Model Domain* phase. Data elicitation is basically finding out the answer to (a set of) questions; organizing these answers, writing them up, placing them in a comparative structure is properly part of *Integrate Data*. The hand-off from

Elicit Data to *Integrate Data* is done through the relatively unstructured means of recording DOMAIN DATA, including:

- video and audio recordings made during an interview or observation session,

- handwritten notes made by the investigator during an analysis, or

- laboratory notebooks kept while collecting data during an experiment or an automated observation.

DOMAIN DEFINING RULES

Rules of inclusion and exclusion for domain membership and the logical relationship between these rules. The rules can be written informally or formally; for example, they could be complex predicates built out of atomic statements composed with logical operators (and, or, etc.) or simple natural language statements. They should distinguish the following categories of rules, however:

- Inclusion vs. exclusion;

- Essential vs. accidental;

- Linkage to exemplars (as described below).

A worksheet template that can be used to cross-validate DOMAIN DEFINING RULES against exemplar systems is shown in Exhibit C-8, EXEMPLARS/DEFINING RULES MATRIX. The DOMAIN DEFINING RULES workproduct is produced during the *Bound Domain* task of the *Plan Domain* phase.

In the *Focus Domain* task of the *Plan Domain* phase the DOMAIN DEFINING RULES are extended by identifying individual features as within or external to the domain scope.

DOMAIN DEFINITION

A composite workproduct consisting of the DOMAIN DESCRIPTION, DOMAIN DEFINING RULES, EXEMPLAR SYSTEMS SELECTION, DOMAIN SETTINGS, DOMAIN CLASSIFICATIONS, DOMAIN INTERACTIONS, and DOMAIN HISTORY. The DOMAIN DEFINITION is created in the *Define Domain* sub-phase in the *Plan Domain* phase.

DOMAIN DESCRIPTION

A refinement of the informal domain description included in the DOMAIN SELECTION, based on insights gained in this task. This serves as a consensus document for the project team about what the domain includes. As such, it is an implicit input to almost every activity downstream in the domain engineering life cycle. It can also be used to orient new project members, in presentation to management and to external audiences, and to communicate the intended scope of the domain of focus to informants during the *Model Domain* phase. The DOMAIN DESCRIPTION is produced in the *Bound Domain* task in the *Plan Domain* phase.

DOMAIN DOSSIER

The evolving "map" of the data sources from which the DESCRIPTIVE MODELS will be derived. The bulk of the dossier will be reports from knowledge elicitation sessions, along with an index to these reports, to facilitate access to them in the *Plan Data Acquisition* task and later during the *Develop descriptive models* sub-phase.

Also included in the DOMAIN DOSSIER is information about the historical or "genealogical" relations between representative systems. Some of this information was elicited as a basis for making the REPRESENTATIVE SYSTEMS SELECTION in the *Plan Data Acquisition* task. Additional information about system genealogy will emerge from the *Integrate Data* task as well.

The DOMAIN DOSSIER workproduct is produced in the *Integrate Data* task in the *Model Domain* phase.

domain engineer

An engineer that participates in one or more of the phases of the domain engineering life cycle. In the *Plan Domain* phase domain engineers may take on the role of domain [engineering project] planner (as distinct from typical project planning); in the *Model Domain* phase the engineer takes on the role of domain modeler; in the *Engineer Asset Base* phase the domain engineer takes on the role of asset base engineer. More specific roles apply within each phase. See also domain planner, domain modeler, asset base engineer.

domain engineering

The creation of new technological and human systems for managing domains within some ORGA-NIZATION CONTEXT, potentially transforming both the organizations and technical systems within that context. Domain engineering includes creating a DOMAIN MODEL, ASSET BASE ARCHITEC-TURE and reusable ASSETS for a particular domain. ODM incorporates a life cycle and process model for domain engineering which is described by the ODM Process Model explained in Part II of this document.

domain engineering initiative

Synonym for domain engineering project.

domain engineering life cycle

The domain engineering life cycle contains processes and workproducts that specifically address domain engineering concerns. These domain engineering processes together form the core ODM process model. This life cycle is distinct from and orthogonal to the system development life cycle. Processes that do not address domain engineering are allocated to sets of supporting methods. See also core ODM process model, supporting methods.

[domain engineering] project

A project that performs an instantiation of the domain engineering life cycle. The project cannot be defined in terms of a project based around a particular domain, since domain selection is part of the life cycle. The project may be part of a larger reuse effort that includes, for example, a system reengineering effort, a corporate reuse planning effort, etc. The project refers in this case specifically to that portion of the larger effort committed to performing the activities within the scope of a domain engineering effort.

domain functionality

An informal term that denotes that portion of functionality within the scope of the domain that occurs within any given exemplar system. Alternative terms that may occur in this document might include "domain system," "domain functional scope," "domain subsystem," "domain instance," "domain instantiation," or "domain-in-system". (This is clearly a term cluster in desperate search for a canonic term!)

domain genealogy

A model that shows historical relationships between representative systems in a domain. In particular, it shows relations relevant to reuse and domain engineering, e.g., leveraged code, requirements reused on new platforms, etc. The classic information captured by such a model would be a

system family relationship. The term genealogy is meant to suggest that more remote "family relationships" can also be important to note.

This information is relevant to three tasks in particular: in *Situate Domain*, in order to develop the DOMAIN HISTORY; in *Plan Data Acquisition*, in order to select a suitably diverse REPRESENTATIVE SYSTEMS SELECTION; and in *Interpret Domain Model*, when information from these two sources is used in order to facilitate correct interpretation of the commonality and variability observed across the representatives. Full creation of what was formerly called a domain genealogy model, while still considered of value to many organizations, is no longer called out as a core workproduct in ODM.

DOMAIN HISTORY

This workproduct situates the domain of focus with respect to *historical* predecessor and anticipated successor systems. Exhibit C-10, CONCEPTUAL/HISTORICAL RELATIONS MATRIX contains a worksheet template that can be used to validate the DOMAIN HISTORY by mapping domains to their evolution. The DOMAIN HISTORY workproduct is produced in the *Situate Domain* task in the *Plan Domain* phase.

<domain> informant

An individual affiliated with a specific stakeholder organization who is accessed as a source of information during the *Model Domain* phase of domain engineering. Informants' knowledge and experience can be characterized in terms of roles they have played within various system settings. For example, a veteran designer could be characterized as having played a role in a number of the design efforts performed by a given company.

Informants may be accessed via direct interaction (interviews, group meetings), via targeted surveys and questionnaires, or indirectly through being the source of an article or document used in data acquisition. In the latter case the informant usually brings some perspective other than that of an individual system.

See also system setting.

DOMAIN INFORMANT KNOWLEDGE

A data item which consists of knowledge elicited from domain practitioners in interviews or via observation and process capture. Knowledge can include knowledge about specific representative systems or more general knowledge about domain concepts, terminology, etc. DOMAIN INFORMANT KNOWLEDGE is a component of the DOMAIN STAKEHOLDER KNOWLEDGE data item.

DOMAIN INFORMATION

A data item containing information about the domain derived from EXEMPLAR SYSTEM ARTIFACTS or DOMAIN STAKEHOLDER KNOWLEDGE. DOMAIN INFORMATION is a component of the ORGANIZATION INFORMATION data item.

DOMAIN INTERACTIONS

This workproduct situates the domain of focus with respect to the *structural* relations outlined above: i.e., the component sub-domains, application settings, and peer domains within operational settings.

This workproduct can be structured similarly to the DOMAIN CLASSIFICATION. However, the rules for how defining rules differentiate the domains do not transfer. For a component sub-domain, identify the high-level defining rules or features that are, effectively, decomposed by the sub-

domain. In theory, any subset of the domain of focus defining rules could be treated as a sub-domain in this sense, but it only makes sense for certain subsets.

For each key interaction of the domain of focus with a related domain, the interface can be characterized in terms of the original domain attributes, particularly encapsulated vs. diffused. Differentiate structural interactions that are well-contained and modularized (a subsystem interface) from those that involve highly interlocked areas of functionality (e.g., calls to memory management, user interface calls, exception handling).

The DOMAIN INTERACTIONS workproduct is produced in the *Situate Domain* task in the *Plan Domain* phase.

domain interconnection

See domain relation.

DOMAIN LEXICON

The DOMAIN LEXICON collates terms drawn from the domain, annotated to reflect information about each term with domain-specific meaning. Terms can be drawn from all domain information sources, including development artifacts (code, design, tests), documentation, literature, and terminology noted in interviews and meetings with domain informants. The DOMAIN LEXICON indicates the domain of focus and might include links, if appropriate, to lexicons for related domains (following relationships in the DOMAIN CLASSIFICATION or DOMAIN INTERACTIONS.

The lexicon consists of a set of entries that group terms with synonymous semantics with respect to the domain scope. The structure of each entry is as follows:

- One or more domain terms which are, by virtue of being clustered within a single entry, declared to be synonymous with respect to the domain.

- Each term in the lexicon has one or more precedent links to some dossier material that shows the "term in use." For example, the link may point to a definition or a reference to the term in an article or its documented use by an informant during an interview.

  Note that this is not the same thing as having a link to an exemplar for the concept captured by the term. For example, if there were a lexicon entry for the term "outline document" the precedent link might point to a user manual where this specific term is used. It need not be pointing to an actual outline document as an artifact studied and noted in the DOMAIN DOSSIER, and if there were such an artifact there would still need to be precedent to the terminology itself.

- A designated canonic term for the set of terms clustered in the entry. This is a standard term to be used in normalized references in lexicon definitions for other terms in the term cluster. This standard term is also the term used in the DESCRIPTIVE MODELS. The canonic term could be one of the terms in the term cluster that is "promoted" to mayoral status; or it could be an adapted term devised by the modelers.

- There may be optional other lexical relationships that link the term to others in the lexicon. Examples of these other relationships include antonyms, oppositional pairs and inverse operations (e.g., "promote" versus "demote" headings in the Outlining domain). In general, these other relations would be determined by a lexical analysis supporting method if one is used.

The DOMAIN LEXICON workproduct is created in the *Develop Lexicon* task in the *Model Domain* phase. See also related domain, precedent, canonic term, term cluster.

DOMAIN MODEL

The final form of the model produced in the *Model Domain* phase. Besides the existing INTER-PRETED DOMAIN MODEL this model includes the following:

- *Extended model of domain features.* Contains both precedented and unprecedented features and innovative extensions to features. The representation form used should be compatible with those used in earlier sub-phases. These features will be a superset of those in the INTER-PRETIVE DOMAIN MODEL.

- *Extended setting characteristics.* Contains semantic information about attributes of potential settings of interest for model features, including "hypothetical" new settings.

The DOMAIN MODEL is produced during the *Resolve Domain Model* task of the *Model Domain* phase. The *Resolve Domain Model* task involves defining numerous connections between features and settings information. The exact partitioning strategy for the DOMAIN MODEL will depend on the supporting methods used.

[domain] modeler

A person performing modeling activities in the context of an ODM project. Strictly, a domain engineer performing activities in the *Model Domain* phase. In usage, sometimes used as a synonym for domain engineer. See also domain engineer.

domain modeling

Development of the DOMAIN MODEL. Also used to denote the general cognitive skill of descriptive domain modeling and taxonomic modeling as differentiated from system modeling.

domain modeling team

Members of the domain engineering project who perform domain modeling. See also domain modeling, domain engineering project.

domain [of focus]

A domain selected for performing domain engineering.

domain of interest

A domain within the scope of interest of stakeholders in a given ORGANIZATION CONTEXT.

domain planning

Another term for the *Plan Domain* phase of the Domain Engineering life cycle.

domain planning team

Members of the domain engineering project who are performing domain planning.

domain practitioner

People who are involved in domain settings, settings where domain-relevant activities is performed. For software domains, these practitioners can include developers, testers, field installers, customer service reps, value-added resellers, maintainers, and, last but not least, end-users of systems. We use the term practitioner to emphasize that people situated in work settings of any kind

are not just agents performing processes, but people who share culture, language, habits, previous experience and tacit knowledge with their co-workers. A network of such people is sometimes called a *community of practice*.

domain relation/interconnection

A relationship between the domain of focus and a related domain. See also analogy domain, specialization sub-domain, component sub-domain, application context. A synonym is domain interconnection.

DOMAIN SELECTION

Documents the domain that was selected as a domain of focus, and the rationale for choosing this domain. The DOMAIN SELECTION includes:

- A characterization of candidate domains in terms of the DOMAIN SELECTION CRITERIA. For each candidate domain, this characterization tells whether the domain satisfies or conflicts with each selection criteria.

- Include the various characterized attributes of the selected domain: native/innovative, vertical/horizontal, encapsulated/diffused, etc.

- The name of the selected domain of focus, along with the rationale for the selection.

- An informal description of the selected domain of focus.

Exhibit C-7, DOMAINS/CRITERIA MATRIX, contains a worksheet template that can be used to evaluate each domain of interest with respect to the identified DOMAIN SELECTION CRITERIA.

DOMAIN SELECTION CRITERIA

A list of prioritized selection criteria that will be used to select the domain of focus, with annotations explaining the derivation of each criterion. Also included is a mapping from domain selection criteria to PROJECT OBJECTIVES to show whether each domain selection criterion supports or conflicts with each objective. Exhibit C-6, PROJECT OBJECTIVES/CRITERIA MATRIX, contains a worksheet template that can be used to map DOMAIN SELECTION CRITERIA to PROJECT OBJECTIVES.

domain setting

Each exemplar system has its own **exemplar system life cycle** of settings which can be related temporally (e.g., development phases) or concurrently (e.g., multiple development sites, multiple user environments). Whereas a particular system will have its particular system settings, it is usually convenient to describe these settings according to a few categories or types. The term **domain settings** distinguishes these generic setting categories from system settings, which would be instances of domain settings. The DOMAIN SETTINGS workproduct included in the DOMAIN DEFINITION will usually encompass only a subset of these domain settings, those which are relevant to the domain hence included in the domain's focus.

More formally, a domain setting is a named category that groups system settings from across a set of exemplar systems according an analogous role these settings play in their respective system life cycles. For example, "development setting" would be a name for a domain setting and for each exemplar system there would be one or more of these settings. Not every domain setting need occur in each exemplar system, and some settings (like usage settings) might have multiple occurrences in a given exemplar system life cycle.

The domain settings applicable for a given domain are specified at a high level in the DOMAIN SETTINGS produced during *Focus Domain*. See also development setting, system setting, usage setting.

DOMAIN SETTINGS

A set of names for particular types of settings that occur in one or more exemplar systems for the domain and which are deemed within the domain scope. The settings are related via links that describe some system life cycle relationship. The domain settings will generally be a subset of all the system settings for the exemplars.

Usually (but not always) this subset will be a contiguous groupings of settings that are upstream or downstream from each other. For example, suppose a a life cycle included the development, customization, field installation and end-user settings. Although the domain has been "bounded" in terms of which systems are exemplars, a domain focusing on development activities would clearly be a "different" domain than one focused on the work processes of the end-users of the system. Either of these could be selected as the domain scope. Or the development and customization settings could be selected as the scope; or the field installation and usage settings. It would, however, be not desirable to focus on the development and usage settings but exclude the customization and field installation settings from the domain scope. This is because one of the purposes of the Model Domain phase will be to model the features of the domain, and to explore ways of shifting certain features across setting boundaries. This means shifting across the intervening settings, and therefore they should be included in the scope of interest.

The DOMAIN SETTINGS workproduct is produced in the *Focus Domain* task in the *Plan Domain* phase.

domain-specific

Semantics particular to the domain of focus; i.e., not specific to an individual system, nor a general part of the representations used. The distinctions are in reality more of a spectrum than a hard and fast line. For example, a model of the Ada data type hierarchy is not specific to a particular application area, but is specific to Ada-based systems.

domain-specific meta-model

Synonym for the INTEGRATED DOMAIN MODEL, which forms a "model of models" for the domain. The INTEGRATED DOMAIN MODEL documents and formalizes relationships and interconnections between the separately developed DESCRIPTIVE MODELS.

domain stakeholder

A stakeholder with interests in the domain models and assets to be produced by the domain engineering project. Some project stakeholders may not be domain stakeholders. For example, a reuse technology developer will generally not be a stakeholder in the selected domain. Once selected, the domain will also define certain stakeholders who may not have been noted as explicit project stakeholders before.

DOMAIN STAKEHOLDER KNOWLEDGE

A data item consisting of knowledge of domain stakeholders about systems and processes in the domain; generally gleaned from informal contact during planning, more formal interviewing in the modeling phase. DOMAIN STAKEHOLDER KNOWLEDGE is a component of the ORGANIZATION INFORMATION data item. This knowledge can also include knowledge not particularly closely tied to any one particular exemplar system; e.g., secondary data sources (survey articles, textbooks, etc.).

DOMAIN STAKEHOLDER MODEL

A refinement of the PROJECT STAKEHOLDER MODEL which only includes stakeholders with an interest in the domain of focus. The DOMAIN STAKEHOLDER MODEL includes any new stakeholders with interest in the domain, based on the informal domain description. The DOMAIN STAKEHOLDER MODEL also includes the specific roles of all stakeholders with respect to the domain of focus. The DOMAIN STAKEHOLDER MODEL is developed during the *Select Domain of Focus* task in the *Plan Domain* phase.

DOMAIN TERMS

Terms that have specific meanings for practitioners in the system settings studied, as deduced from the artifacts and informant data. The domain terms are linked to their occurrence in particular elicitation sessions. Syntactic relations such as synonyms are noted only where they are clearly suggested by the data in the domain or directly suggested by informants.

The DOMAIN TERMS are collected during the *Integrate Data* task during the *Model Domain* phase. Terms are not standardized in this task; this is done in the *Develop Lexicon* task within the subsequent *Describe Domain* sub-phase when the DOMAIN LEXICON is developed. See also DOMAIN LEXICON.

DOMAINS/CRITERIA MATRIX

A worksheet that can be used to evaluate each domain of interest with respect to the identified DOMAIN SELECTION CRITERIA. A template of this worksheet is illustrated in Exhibit C-7. The DOMAINS/CRITERIA MATRIX worksheet can be used in developing the DOMAIN SELECTION workproduct, during the *Select Domain of Focus* task in the *Plan Domain* phase.

DOMAINS OF INTEREST

The DOMAINS OF INTEREST workproduct includes:

- A list of domains, annotated with attributes for each domain, with definitions and values for each attribute.

- A many-to-many mapping of the intersection of DOMAINS OF INTEREST with systems of interest. This mapping characterizes domains by their exemplar systems.

- A many-to-many mapping of DOMAINS OF INTEREST to stakeholders with interests in systems that intersect with the domains.

Exhibit C-5, DOMAINS/SYSTEMS MATRIX, shows a worksheet template that can be used to map the intersection of DOMAINS OF INTEREST with systems of interest. A similar format can be used to map the intersection of DOMAINS OF INTEREST with candidate project stakeholders.

Purposes of the DOMAINS OF INTEREST workproduct include:

- Candidate domains in the DOMAINS OF INTEREST are input to selecting the domain of focus.

- Input to the *Define Domain* sub-phase, used in developing the DOMAIN DEFINITION.

- Reused in subsequent domain engineering projects as a starting point for domain selection. If the ORGANIZATION CONTEXT is stable, the domains of interest to the organization should change relatively slowly (e.g., only if workers develop expertise in new areas, or if existing areas are partitioned into new domains).

DOMAINS/SYSTEMS MATRIX

The DOMAINS/SYSTEMS MATRIX worksheet can be used to map the intersection of DOMAINS OF INTEREST with systems of interest, during the *Characterize Domains of Interest* task in the *Plan*

Domain phase. The worksheet contains a many-to-many mapping of DOMAINS OF INTEREST to systems of interest. A system is an exemplar of a domain if the domain functionality occurs within the system scope. The DOMAINS/SYSTEMS MATRIX characterizes domains by their exemplar systems. A template of the DOMAINS/SYSTEMS MATRIX is illustrated in Exhibit C-5.

economies of scope (non-ODM term)

A market analysis technique used in reuse economic analysis

encapsulated domain

A horizontal domain where functionality is clustered in one structural component of the system in question. See also horizontal domain.

engineering

See asset base engineering.

engineering asset

Engineering assets include the traditional notion of reusable components (possibly spanning the engineering life cycle) that will be retrieved by application engineers, possibly tailored or modified, and then incorporated into their own products. Some comparative sampling of engineering artifacts of various types should have been done as part of the *Acquire Domain Information* subphase of the *Model Domain* phase. This information, available in the DOMAIN DOSSIER, can help asset developers identify candidate artifacts for reengineering.

Other artifacts that are candidates for transformation into assets are interim artifacts used in system development but not part of the deliverable system. Examples include test data, debugging scripts, requirements checklists used by analysts as reminders, etc. See also asset.

essential [feature]

A required feature that applies to all exemplar systems in the domain, even those not examined so far. See also accidental.

ethnographic technique

A technique for gathering information from people via face-to-face interview, observation, participant observation or other relatively non-invasive methods that preserve some of the contextual information otherwise easily passed over. Described in detail in the Information Acquisition Techniques supporting method in Section 8.2.

exclusion [rule]

A defining rule that describes a feature which does *not* hold for exemplar systems.

exemplar

See exemplar system. An informal term that can be ambiguous, since exemplar artifacts are also frequently referred to. Exemplar system is the preferred term.

exemplar set

The set of individuals considered and determined to be embraced by the concept. The exemplar set is always a finite subset of the extent. See also extent.

exemplar system

A system or subsystem in the scope of which domain functionality occurs. Exemplar systems are enumerated in the EXEMPLAR SYSTEM SELECTION. A system is an exemplar of a domain if the domain functionality occurs within the system scope.

NOTE: The term "exemplar system" is something of a misnomer. In the case of a domain that is vertical with respect to a given exemplar system, the entire system is considered within the domain scope; the domain is scoped in terms of sets and subsets of systems. For horizontal domains, or domains that include slices of functionality within larger applications, use of the term "exemplar system" is ambiguous. It can refer to only the domain functionality where it intersects the system, or it can refer to the system as a whole in some cases. Since some domain information gathering activities are planned at the level of overall systems (e.g., identifying experts and other informants for the system, obtaining documentation) this proves a useful term. When the intent is simply to denote the system, subsystem or embedded functionality pertaining to the domain, the less restrictive term "exemplar" by itself is preferable.

See also counter-exemplar, borderline exemplar, core exemplar

exemplar system architecture

The system architecture of an exemplar system. See also system architecture.

EXEMPLAR SYSTEM ARTIFACTS

A data item consisting of artifacts drawn from any phase of the life cycle, from an exemplar system for the domain.

exemplar [system] setting

A system setting for an exemplar system, where the setting itself is of interest to the domain engineering project. Let system S be an exemplar of domain D with set of system settings S1 through Sn. Only certain system settings will be of interest with respect to D; namely, those that correspond to a domain setting in the DOMAIN SETTINGS. These are exemplar system settings: that is, the system is an exemplar system with respect to the domain, and the system setting is one relevant to the domain. See also system setting, exemplar system life cycle.

exemplar system life cycle

The settings in which a system is developed and used, together with any other settings for the system and the interconnections among the settings. The settings can be related temporally (e.g., development phases) or concurrently (e.g., multiple development sites, multiple user environments). In general, therefore, the life cycle is a network of settings, rather than a linear sequence. See also exemplar system setting.

EXEMPLAR SYSTEMS SELECTION

A list of exemplar systems for the domain that complements the DOMAIN DEFINING RULES by identifying systems judged to be members of the domain. The EXEMPLAR SYSTEMS SELECTION should document *all* systems that planners have considered with respect to domain membership. A worksheet template that can be used to cross-validate DOMAIN DEFINING RULES against exemplar systems is shown in Exhibit C-8, EXEMPLARS/DEFINING RULES MATRIX. Exemplar systems are chosen from CANDIDATE EXEMPLAR SYSTEMS in the *Bound Domain* task of the *Plan Domain* phase. See also exemplar system, counter-exemplar, borderline exemplar, CANDIDATE EXEMPLAR SYSTEMS.

EXEMPLARS/DEFINING RULES MATRIX

A worksheet template that can be used to cross-validate DOMAIN DEFINING RULES against exemplar systems is shown in Exhibit C-8, EXEMPLARS/DEFINING RULES MATRIX.

A worksheet that can be used to cross-validate DOMAIN DEFINING RULES against exemplar systems selected in the EXEMPLAR SYSTEMS SELECTION. A template for the EXEMPLARS/DEFINING RULES MATRIX worksheet is illustrated in Exhibit C-8. The EXEMPLARS/DEFINING RULES MATRIX worksheet is used during the *Bound Domain* task in the *Plan Domain* phase.

extensional definition

A term borrowed from the field of logic. Defining a category by enumerating examples to suggest what membership means. Extensional definition involves designating a set of systems as **exemplars** of the domain. These are documented in the EXEMPLAR SYSTEMS SELECTION. The extensional definition provides concrete examples that everyone can relate to as a common basis for understanding, hence offers empirical validation of a shared understanding of the domain. Used in conjunction with intensional definition methods of defining rules. See also intensional definition.

extent

The set of all possible individuals embraced by a concept definition.

external architecture constraint

An Architecture constraint which is imposed by those portions of potential target applications (and their surrounding environments) that lie outside the domain. These constraints impact the variability of the interfaces to and from the external functionality which the asset base must accommodate. See also internal architecture constraints.

EXTERNAL ARCHITECTURE CONSTRAINTS

This workproduct includes the following information:

- A list of the key external interfaces to (and from) ASSET BASE functionality which the ASSET BASE must support, for each customer setting for which reusable ASSETS are to be developed. Interfaces include environmental and sub-system interfaces. Environmental and sub-system interfaces can be partitioned if appropriate, but it is possible that the determination of which category an interface falls into may depend on final architectural decisions that will be decided in a later task.

- An allocation of features from the ASSET BASE MODEL to the external interfaces relevant to each customer setting. The FEATURE/EXTERNAL INTERFACE MATRIX worksheet template in Exhibit C-14 illustrates one way in which this information can be expressed.

- Documentation of the relevant constraints each customer in the ASSET BASE MODEL has on each external interface. The CUSTOMER/EXTERNAL INTERFACE MATRIX worksheet template in Exhibit C-15 illustrates one way in which this information can be expressed.

The EXTERNAL ARCHITECTURE CONSTRAINTS workproduct is developed in the *Determine External Architecture Constraints* task in the *Engineer Asset Base* phase.

feasibility

The technical difficulty and cost associated with providing a feature or feature set to candidate customers. See also utility.

feature

A statement or assertion associated with an exemplar system workproduct. A feature is used to express a differentiating behavior or characteristic associated with a system artifact that is significant to some domain practitioner. Each feature is true or not true for each exemplar system.

The term can also be used to denote an entire **feature model** describing some aspect of variability for a system, artifact, etc. In its simplest form, a feature model includes a feature category and various feature variants or feature values. In domain modeling, **defining features** for the domain are features that hold for all exemplars in the domain. The defining feature would typically become the top-level feature category of a feature model, and differentiating features (things that make one exemplar different from another with respect to the feature category) would be the feature variants for that model. See also defining feature, differentiating feature, feature variant, feature model.

feature binding site

Points in the workflow at which feature variants may be selected. Depending on the domain setting and the feature binding site, this selection may be done programmatically, by software, or by human decision-making. Different representative systems will make different choices with regard to where feature variants are bound. Some systems will have sites that do not occur elsewhere. Some will provide multiple options.

feature cluster

Pattern of features that consistently co-occur (or consistently *never* co-occur) across exemplars. These include features that co-occur in several systems, as well as feature constraints, combinations of features that seem to rarely co-occur.

feature combination

Synonym for feature set. Used in the *Resolve Domain Model* task to denote feature sets that are interim constructs generated from applications of transformation rules; feature set is used later, in the *Correlate Features and Customers* task, to denote sets that have been judged as coherent and stable as a whole. See also feature set.

feature configuration

Synonym for feature set.

feature constraints

Combinations of features that seem to rarely occur. See also feature cluster.

FEATURE/CUSTOMER MATRIX

A worksheet which can be used to develop a correlation (or mapping) between customers/settings and candidate feature sets, reflecting the business scenarios, in the CUSTOMER DOSSIER, ASSET BASE DOSSIER, and ASSET BASE MODEL workproducts. The FEATURE/CUSTOMER MATRIX worksheet is first developed during the *Correlate Features and Customers* task in the *Engineer Asset Base* phase. The mapping between customers and settings shows the anticipated degree of interest that the candidate customers (in their relevant settings) have for the candidate feature sets. The mapping is *many-to-many*: Each feature set may be deemed of interest to multiple customers. Conversely, each customer may be interested in multiple feature sets.

- For feature sets that are components of the domain, the more components that interest a given customer the more likely it is that the customer will utilize multiple assets in sub-system level

435

aggregations.

- For feature sets that are specializations/generalizations of the domain, a customer's interest in more than one feature set indicates the degree of variability that is required in the ASSET BASE.

In the *Prioritize Features and Customers* task in the *Engineer Asset Base* phase, the FEATURE/ CUSTOMER MATRIX worksheet can be refined, as part of the ASSET BASE DOSSIER, to contain a mapping between the feature sets and the customer settings to which they apply, prioritized in terms of key criteria such as utility and feasibility.

In the Select Features and Customers task in the *Engineer Asset Base* phase, the FEATURE/CUS-TOMER MATRIX worksheet can be refined, as part of the ASSET BASE MODEL, to contain a mapping of selected features to selected customers and settings.

The FEATURE/CUSTOMER MATRIX worksheet template is shown in Exhibit C-13.

feature distance

A metric for how much semantic distance exists between two feature variants in a model.

feature ensemble

A feature set that can be realized as a single implementation made available within a system setting. This could be the feature profile for an entire system instance within the domain (i.e., an architectural variant) or an ensemble to be implemented by multiple components made available as a library (e.g., a set of system services) within a development setting.

FEATURE/EXTERNAL INTERFACE MATRIX

Contains an allocation of features from the ASSET BASE MODEL to the external interfaces relevant to each customer setting. A template of this worksheet is illustrated in Exhibit C-14. The FEA-TURE/EXTERNAL INTERFACE MATRIX worksheet can be used in developing the EXTERNAL ARCHI-TECTURE CONSTRAINTS workproduct, during the *Determine External Architecture Constraints* task in the *Engineer Asset Base* phase.

feature model

A model of the distinguishing characteristics of specific artifacts or representative systems as a whole. Each feature is an assertion or statement that holds for the associated system entity and indicates a significant differentiation from the standpoint of some domain practitioner.

Each feature model maintains the following linkages:

- The stakeholder(s) for whom this feature is relevant or "of interest." This will usually be a domain practitioner in some domain setting, e.g., a system user in the usage setting, a software implementor in the development setting.

- The setting in which the feature is relevant. (Usually strongly correlated with the relevant stakeholders, as above.)

- The "feature category" that encompasses the observed variants.

- The "feature values" that represent the variability with respect to that feature within the domain scope.

See also feature, FEATURE MODELS,

FEATURE MODELS

Models of distinguishing characteristics of specific artifacts or representative systems as a whole. The FEATURE MODELS are developed in the *Model Features* task in the *Model Domain* phase. See also feature, feature model.

feature of interest

A short, informal sentence describing features of a system or many systems to which attention was drawn during data elicitation. Each short description is annotated with its sources and the setting to which it applies.

feature model

A model describing some aspect of variability for a system, artifact, etc. In its simplest form, a feature model includes a feature category and various feature variants or feature values. In domain modeling, **defining features** for the domain are features that hold for all exemplars in the domain. The defining feature would typically become the top-level feature category of a feature model, and differentiating features (things that make one exemplar different from another with respect to the feature category) would be the feature variants for that model.

More formally, a model F is a feature model for another (concept) model C if there is an interpretation from the elements of F to features (assertions or properties) that differentiate the concepts in C.

feature profile

Generally, a profile allocates a set of features to any entity to be characterized in terms of those features. A set of feature statements that hold for a particular artifact or asset. Feature profiles map specific features or feature combinations onto domain settings (e.g., usage, development) to which they may apply. A **descriptive feature profile** is a **feature set** that characterizes a particular representative system. Descriptive feature profiles are created after feature sets are combined into the INTEGRATED DOMAIN MODEL. A **prescriptive feature profile** is a feature set that characterizes a specific target system for the asset base; or it can be allocated to an asset specification, part of the ASSET BASE ARCHITECTURE or later ASSET IMPLEMENTATION PLAN.

feature set

A feature set can be any collection of features within a FEATURE MODEL. In the *Resolve Domain Model* task feature sets are identified that satisfy certain formal properties of closure, cohesion and orthogonality. Each of these feature sets is associated with settings in which the feature set could be useful to practitioners; these are the potential market profiles for the feature set. See also feature profile.

feature space

Denotes the range of feature combinations possible for a given set of feature variants.

feature variant

An immediate specialization of a feature within a feature model. An arbitrary descendant is a **feature specialization**. Two features are variants if they have been analyzed as being specializations of a common more general feature.

FEATURES OF INTEREST

Short, informal sentences, each describing features of a system or many systems to which attention was drawn during data elicitation. Each short description is annotated with its sources and the setting to which it applies. The FEATURES OF INTEREST workproduct is produced in the *Integrate Data* task in the *Model Domain* phase.

flexible architecture

See highly flexible architecture.

formal feature

An informal feature statement in which each key domain-specific term or phrase is linked to elements in the CONCEPT MODELS. (not currently used)

[concept modeling] formalism

One component of a Concept Modeling supporting method; the theory or formal system that provides a strict semantic interpretation to a concept modeling representation. (There are, of course, other kinds of formalisms, but these are the ones relevant for domain modeling.)

generalization

An entity generalizes another entity when it has *fewer* features or attributes, and/or addresses a broader context of application. See also specialization.

generalization domain

A domain D1 is said to *generalize* a domain D2 if:

- The exemplar set, or ORGANIZATION CONTEXT, of D1 is a superset of the exemplar set of D2;

- The set of defining features for D1 is a *subset* of the defining features for D2 (i.e., a less tightly constrained definition).

generative asset

See generative technology.

generative implementation

See generative technology.

generative technology

Use of programming techniques that allow implementations to be generated from a high-level specification. In the ODM context, the specification is typically a domain-specific specification utilizing terms from the lexicon and/or FEATURE MODELS.

generic architecture

A system architecture which generally has a fixed topology but supports component plug-and-play relative to a fixed or perhaps somewhat variable set of interfaces. A generic architecture does not satisfy the needs of an ODM asset base architecture in general, although it may suffice in some circumstances. Generic architectures, given the current state of practice, do not readily support the degree of variability that is often needed to satisfy diverse customer settings (e.g., topo-

logical variations, robust architectural subsetting, generic subsystem-level architectures adapting to diverse system architectures).

heterogenous domain

A domain with diverse structural elements such as different architectures across exemplars. The conventional domain view favors large-scale domains with common architectures. Associated with this approach is the notion that the purpose of domain modeling is to capture only the commonality within the domain. Domains with diverse structure, i.e., heterogenous domains, generally are not handled well by such methods. They are heartily welcomed in ODM, where the goal is very tightly constrained range balanced by as rich diversity as possible within the boundaries. They are distinct from **diffused domains**, which are generally **horizontal** in scope with respect to their **exemplar systems**. See also homogenous domain.

highly variable architecture

An architecture that supports variability by replacing or masking components, but via structural variation in the topology of the architecture itself. In particular, highly variable architectures may need to provide both optimized and non-optimized versions of particular component configurations, simultaneous access to multiple structural layers for certain configurations, etc.

homogenous domain

A domain with common and somewhat uniform internal structure. See heterogenous domain.

homonym

One of two or more words that are spelled and pronounced alike but have different meanings. Used as a defined relation between terms in the DOMAIN LEXICON.

horizontal domain

A domain that includes only a subset of the functionality implemented in a system. Horizontal domains may be encapsulated or diffused. In ODM the terms *vertical* and *horizontal* are not absolute qualities of general functional areas, but describe the span of a given domain's coverage relative to a particular set of systems. For example, within a family of large-scale systems, database management may be considered a horizontal domain. In the Database Management Systems provider marketplace, database management could be considered a vertical domain. See also vertical domain, encapsulated domain, diffused domain.

inclusion [rule]

A statement about various combinations of features that hold for exemplar systems in the domain.

informant

Synonym for domain informant.

information

In the context of ODM, data that is acquired through a process of elicitation. Organization information is the most general, domain information specific to the domain of focus.

439

information source

A specific document or informant used as a source of information for domain engineering. Information sources are instances of one or more information types.

INFRASTRUCTURE IMPLEMENTATION PLAN

A plan for implementing the ASSET BASE INFRASTRUCTURE. The INFRASTRUCTURE IMPLEMENTATION PLAN includes technology selections and scheduling constraints for infrastructure development relative to development of ASSET releases. The infrastructure may be specifically developed for the ASSET BASE or may be based on available technology that is applied to the ASSET BASE. In general, individual ASSET IMPLEMENTATION PLANS will be affected by the choice of ASSET BASE INFRASTRUCTURE. The INFRASTRUCTURE IMPLEMENTATION PLAN is created in the *Plan Asset Base Implementation* task in the *Engineer Asset Base* phase.

innovative feature

A feature that was generated by innovative thinking or by performing a transformation on a feature model, rather than by examination of domain data. The innovative feature need not correspond to any particular representative system.

Processes throughout the ODM life cycle are intended to spur innovative thinking about domain functionality in a managed way. Innovative ideas for features will therefore be a natural by-product. Features that seem relevant to the domain, but for which there is no precedent in exemplar systems, can be handled in two ways:

- Expand the EXEMPLAR SYSTEMS SELECTION to include new systems exhibiting the features at issue; this may in turn necessitate adjusting the REPRESENTATIVE SYSTEMS SELECTION.

- Document innovative features separately from precedented features, as input to the final task of the *Model Domain* phase, *Resolve Domain Model.*

innovative modeling

The *Resolve Domain Model* task involves application of a repertoire of innovative modeling techniques. Once domain information has been transformed into a formal model representation, it is possible to apply transformations and operations directly on this model data in order to derive new domain possibilities. Such transformations can even be applied by those without extensive domain knowledge; in fact in some respects it may be easier for "domain-naive" modelers because they have fewer pre-conceptions about "what makes sense" in the domain. Of course, in the end domain practitioners must evaluate what is useful and feasible for the domain; but in the innovative modeling process it is helpful to defer these considerations. The techniques rely on the knowledge created during the *Describe Domain* sub-phase and the *Interpret Domain Model* task. In particular, we are looking for innovations that we can get "almost for free":

- By viewing the overall functionality in the domain as a whole, it is often possible to see capabilities that do not require solving new problems, but merely re-packaging or re-configuring solutions that already exist.

- Sometimes, new capabilities arise in fact by *removing* functionality as well as adding it. It is often the "extra work" that a component performs that ties it to particular contextual sets of assumptions and prevents its ease of reuse in new environments.

- New capabilities can also be derived by placing existing capabilities in new settings. Innovation lies in the combination of features with settings, and new discoveries on either side of the equation can lead to important new value creation.

Taking the time to investigate these synergistic opportunities creates significant added value for the overall domain modeling process (as compared to, for example, a parallel set of system devel-

opment efforts). Innovative modeling provides an opportunity for innovative design and the discovery of novel opportunities, through combinations of features that have all been proved individually to be attainable. It can help reveal unexpected commonality across systems and even across domains.

instance

A specific individual of a given type.

INTEGRATED DOMAIN MODEL

A workproduct which contains a unified and integrated model of feature variants synthesized from the individual FEATURE MODELS during the *Integrate Descriptive Models* task in the *Model Domain* phase. This model includes:

- *Feature binding sites*. Integrates the FEATURE MODELS with DOMAIN SETTINGS. Each feature is associated with a specific site in a domain setting. In addition, domain practitioners who play various roles in those settings have various interests in and visibility to the features. In this task these relationships are refined to a next level of granularity, to individual feature binding sites within each domain setting and their relationships to practitioner roles.

- *Integrated descriptive model*. A linked version of the separate CONCEPT MODELS and FEATURE MODELS with redundancies and inconsistencies removed.

- *Representative Systems Feature Profile*. A classification of each representative system in the REPRESENTATIVE SYSTEMS SELECTION in terms of the features in the integrated descriptive model.

- *Domain Model Interconnections*. A **meta-model** that shows the relationships between the descriptive models developed.

The INTEGRATED DOMAIN MODEL is developed in the *Integrate Descriptive Models* task in the *Model Domain* phase.

intensional definition

An intensional definition expresses inherent attributes of the domain, not merely features that can be observed of a particular set of exemplars but might be accidentally shared attributes, not relevant to the domain definition.

> Example. If all the exemplars examined for outlining programs ran on the Macintosh, this would not necessarily make the feature "runs on Macintosh platform" part of the intensional definition for the domain. The test to apply is the following: if an exemplar were examined that matched all other features but the one in question, would it be an exemplar or not? If the answer is yes, the feature is extensional, an observed commonality of the exemplar set but not fundamental to the domain definition. If the answer is no, the feature is an intensional, defining feature.

intent

The minimal set of properties that provide a decision procedure for new individual descriptions to determine whether or not they are embraced by the concept. Just as we can only get at the extent through the exemplar set, the intension can only be approximated by an articulated set of defining rules.

441

interest

A strategic relation or other motivation of a stakeholder, in a system, a project objective or a domain. Interests can be both positive (e.g., perceived opportunities or incentives) or negative (e.g., perceived risks, threats or other disincentives).

internal architecture constraint

An architecture constraint which reflect feature interrelationships within the domain. The fact that subsets of assets may be used as distinct collections in different customer settings introduces a dramatic increase in the number of potential variant structures to be considered. This is not the case in engineering for a single system, where the structure has impact primarily on maintainability and related engineering factors. See also external architecture constraint.

INTERNAL ARCHITECTURE CONSTRAINTS

This workproduct includes the following information:

- Identification of which feature ensembles are *desirable* in usage settings and which feature ensembles are *feasible* in development settings. Refines the information in the ASSET BASE MODEL.

- Key constraints pertaining to feature ensembles, such as whether they can be selected as stand-alone or separately selectable. The COMPONENT CONSTRAINTS TABLE worksheet template in Exhibit C-16 illustrates how this information can be expressed.

- Constraints and relationships between subsets/layered architecture variants and feature ensembles. The LAYERING CONSTRAINTS TABLE worksheet template in Exhibit C-17 illustrates how this information can be expressed.

The INTERNAL ARCHITECTURE CONSTRAINTS workproduct is developed in the *Determine Internal Architecture Constraints* task in the *Engineer Asset Base* phase.

INTERPRETED DOMAIN MODEL

An extension of the INTEGRATED DOMAIN MODEL adding:

- Feature clusters with strong rationale for co-occurrence.

- Feature constraints, which reduce the possible feature space to exclude combinations interpreted as out of scope or semantically meaningless.

- Feature occurrences documented as historical "vestiges" rather than strongly correlated with contextual profiles.

- Attributes of domain practitioners or surrounding system settings that are relevant to the explanatory model of features derived in this task.

- Documented process information, rationale, decision histories, trade-offs, and issues surrounding elements in the INTEGRATED DOMAIN MODEL.

The INTERPRETED DOMAIN MODEL is developed in the *Interpret Domain Model* task of the *Model Domain* phase.

interpreted feature

A descriptive feature for which rationale and/or process and contextual information has been obtained in the *Interpret Domain Model* task. Generally the interpreted feature has been correlated with some attributes of the representative system settings in which it occurs as rationale.

investigator

Term for a person collecting information for information in the *Acquire Domain Information* sub-phase.

key exemplar

Key exemplars are systems required to be covered by the domain definition for strategic reasons, typically due to the interests of some key stakeholder, i.e., "If we're going to focus on this domain then we have to consider X as an exemplar." They may not have strong correspondences with principles for inclusions or exclusion beyond their individual case. The term "key" here refers to the strategic or stakeholder-driven importance of these exemplars in the definition process. Key exemplars are distinctive because if defining rules wind up excluding them, the rules must change!

key stakeholder

A stakeholder whose buy-in and commitment are deemed essential to the overall success of the domain engineering project.

knowledge

Don't worry; we aren't really going to try to answer the question "What is knowledge?" in this lexicon (or this document)! In the ODM process model, however, **knowledge** generally refers to information which is actively elicited as part of the modeling process itself, rather than passively received via documentation, i.e., **information**. Also, information is *about* a topic, knowledge is *received from* a source: i.e., ORGANIZATION INFORMATION is information about the organization, which may be obtained from various artifacts and documents or from STAKEHOLDER KNOWL-EDGE. **Stakeholder knowledge** is the knowledge stakeholders have, elicited via interviews, project participation, etc. Generally, when stakeholder knowledge is obtained it has a strategic role in the process; when informant knowledge is obtained it has more of a descriptive or informational role in the process. Stakeholder decisions are not knowledge; hence, processes intended to obtain commitments from stakeholders will not necessarily show STAKEHOLDER KNOWLEDGE as an input. See also information, informant, stakeholder knowledge.

layer

Optional extension to the core ODM process model. Layers have a pervasive impact across the entire domain engineering life cycle. See Section 9.0 of this document for a description of some of the layers that can be added to the core ODM process model.

LAYERING CONSTRAINTS TABLE

Contains constraints and relationships between subsets/layered architecture variants and feature ensembles. A template of a LAYERING CONSTRAINTS TABLE worksheet is illustrated in Exhibit C-17. The LAYERING CONSTRAINTS TABLE worksheet can be used in developing the INTERNAL ARCHITECTURE CONSTRAINTS workproduct, during the *Determine Internal Architecture Constraints* task in the *Engineer Asset Base* phase.

learning asset

Closely related to process assets, learning assets are structured around the roles, profiles, or experience of particular stakeholders. Since a given stakeholder may utilize multiple workproducts and perform many processes, opportunities for encapsulating reusable knowledge into assets may emerge independently of artifact or process structure. For example, automated or codified expert knowledge could be used in decision making or workproduct transformation. Some of the prod-

ucts of domain engineering (like the DOMAIN MODEL itself) can be considered learning assets. See also asset, process asset.

legacy model

If a domain is sufficiently interesting that it has been selected for data acquisition and domain model, it is often the case that some modeling has been done before. The results of previous domain modeling efforts, called legacy models, can appear in a number of guises, including survey articles, consumer reports, or just private theories from particular informants. These pose a particular challenge to the present domain engineering effort, since they come from well-informed practitioners (and hence should not be ignored), but they also move well beyond the goals of data acquisition, in that they have already modeled commonality and/or variability in the domain. How can we give such models the credit they are due, without simply copying them verbatim into the domain model?

There is no magic formula for determining how to incorporate information from such a model into the current dossier. However, the key to understanding the relationship between these *legacy models* and the current domain modeling project is to realize that even if they were not done using ODM, they are nevertheless a product of a particular set of stakeholder interests and biases, in exactly the same way that the current domain selection is a result of the analysis that went on in the *Plan Domain* phase. Hence, legacy models are analyzed in the *Integrate Data* task in order to understand what part of the legacy model is relevant to the current effort and to consider how it relates to the current effort.

lexicon

See DOMAIN LEXICON.

life cycle

The top level, or "root" node, of the ODM process tree (i.e., *Domain Engineering*), representing the overall ODM domain engineering life cycle. See also domain engineering.

market

See customer setting.

mask

A transformation where the entity in question is removed from the system of which it is a part, resulting in a variant structure which is then analyzed for utility and feasibility. This technique can be used on feature sets in the *Resolve Domain Model* task and later, in the *Determine Internal Architecture Constraints* task in the *Architect Asset Base* sub-phase.

meta-model

A meta-model is a model of the relationships and interconnections between other models.

model

Within the ODM context, a model implies one of the domain models as opposed to a system model. These domain models are maps of commonality and variability across systems rather than models of the structure or function of a single system. Models are represented using a Taxonomic Modeling Support Method (See Section 8.3). See DOMAIN MODELS.

model-based development

Related to domain engineering; in the ODM context, system engineering performed on the basis of a DOMAIN MODEL but not necessarily involving the development of an ASSET BASE MODEL, ASSET BASE ARCHITECTURE, or ASSETS. In these cases, the DOMAIN MODEL is basically being used as a comprehensive superset of requirements in the domain (although requirements assets *per se* have not been engineered. This definition may differ significantly from use of the term in other research.

model formalism

The formal language in which a model is written. See Taxonomic Modeling supporting method in Section 8.4.

modeling

See domain modeling.

modeling team

See domain modeling team.

NEW DOMAIN TERMS

Terms that are introduced as a result of the concept modeling activities for which there are not good existing terms in use by domain practitioners. These terms will generally reflect names of concepts that have been introduced in some CONCEPT MODEL or FEATURE MODEL. The terms are passed to the *Develop Lexicon* task to be added to the LEXICON MODEL. NEW DOMAIN TERMS are determined in the *Model Concepts* and *Model Features* tasks in the Model Domain phase.

NEW SYSTEM ARTIFACTS

Any workproducts for representative systems that are prototyped or reverse engineered to fill gaps in the found data. Typically these workproducts will have some other use, but would not have been created if not for their additional value as sources of domain information. Because they are created as data for domain engineering, rather than to build any individual systems, they are considered *artifacts* rather than workproducts. Because these artifacts are typically associated with a single representative system, rather than models of the full domain scope, they are termed "*system* artifacts." Because the artifacts are created in the course of domain engineering, rather than being pre-existing artifacts examined by domain engineers they are "*new* system artifacts." NEW SYSTEM ARTIFACTS can be by-products of the *Elicit Data* task in the *Model Domain* phase. See also artifact, system artifact.

organization

A company, institution, informal group, community group, etc. Some group of people who voluntarily self-identify as a larger entity for some purpose (business, cultural, social, etc.). See also stakeholder context.

ORGANIZATION CONTEXT

A data item which consists of the business strategies, policies and procedures, expertise, technological capabilities, cultural legacies, etc., of the set of organizations involved in a reuse effort.

organization domain

A shared abstraction typically intersecting, and integrating, organizational groups and structures, areas of expertise, and projects oriented around particular software products or systems. An organization domain usually spans several systems, may include only sub-portions of these systems, and will usually map across functional divisions within or even across organizations as well.

ORGANIZATION INFORMATION

A data item which is one of the primary inputs to domain engineering. General sources of ORGANIZATION INFORMATION may include: organization charts, business plans, marketing literature, annual reports, and informal knowledge of project members and domain experts. Other valuable sources of information are results of reuse-specific and broader organization assessments.

organization setting

A set of practitioners in an organization performing activities, interacting with each other, using tools.

Settings can include sub-settings. There can also be interactions across settings (e.g., a tool developed in one setting can be used in another setting) drawing useful boundaries around a particular setting within an organization is a fundamental task in analysis. The way we define organization settings influences the way we define system settings. See also system setting.

organon

A historical term that means "a structured body of knowledge". In the context of this document, a longer-term vision of an asset base, founded on the notion of repositories of codified domain knowledge, coupled with pro-active technologies to support the use and evolution of the knowledge. The organon also includes the community of stakeholders who collaboratively define the domain, build the domain model and engineer and use the assets.

partitioning

Where this term is used, it implies that sub-divisions are non-overlapping and exhaustive.

peer domain

Functional areas that occur at the same level as domain functionality within exemplar systems. A diagram of domain of focus relations with peer domains would most closely resemble a conventional system context diagram or high-level system architecture. Such a diagram would typically depict the domain of focus as a subsystem, with interfaces illustrated to all related subsystems within exemplar systems for the domain. Peer domains abstract these related subsystems and represent the anticipated range of variability to be encountered in those subsystems across multiple exemplars.

phase

One of the three major components of the ODM life cycle: *Plan Domain, Model Domain,* or *Engineer Asset Base.* These are also referred to as the <Domain> Planning, <Domain> Modeling, and <Asset Base> Engineering phases.

planning

See domain planning.

planning team

See domain planning team.

populate

To add descriptions of exemplar systems or artifacts into a model in the appropriate categories as instances. This is distinct from adding a new category or relationship to the model itself. See also instance.

potential

Anticipated at a future point in the ODM process, but not at a point where the decision is currently being made. For example, in the *Plan Domain* phase potential customers may be identified who will not be selected until the *Engineer Asset Base* phase. Once the selection process is initiated (in *Scope Asset Base* sub-phase) candidate customers are considered. See also candidate.

potential market profile

Attributes of an application context identified in the *Resolve Domain Model* task and associated with a particular feature set. For example, an outline browser supporting read-only browse mode might be of interest in an environment where collaborative document preparation was being performed. The latter attribute becomes an attribute of the potential market profile.

precedence

See precedent.

precedent

An instance within an EXEMPLAR SYSTEM ARTIFACT that supports the inclusion of a term or a concept in the *Domain Lexicon* or a DOMAIN MODEL.

prescriptive modeling

In order to emphasize the particular "cognitive style" necessary for domain engineering, the ODM process model separates the life cycle into two distinct phases: a *descriptive* domain modeling phase, and a *prescriptive* asset base engineering phase. The shift from descriptive to prescriptive modeling takes place at the beginning of the Engineer Asset Base phase. In prescriptive modeling binding decisions and commitments about the scope, architecture and implementation of the asset base are made. The descriptive DOMAIN MODEL is transformed into a prescriptive set of committed features for implementation. The prescriptive phase begins by re-scoping the range of functionality to be supported by the reusable assets to be developed; commitments are made to a real set of customers for the asset base. These commitments cannot be finalized in the initial project planning step, because they depend on data that emerge from the descriptive modeling phase. This scope is documented in the Asset Base Model which therefore represents a subsetting of the final extended domain model.

This prescriptive model leads to the development of an ASSET BASE ARCHITECTURE to support the prescribed variants. Prescriptive features are mapped onto the structure of the asset base and to sets of specifications for particular assets. By preserving traceability from features back to exemplar artifacts, asset developers can gain access to potential prototypes on which to base development of new assets. In addition, because assets are described within the asset base in terms of the feature model, asset utilizers can eventually retrieve components based on the same descriptive feature language. See also descriptive modeling.

primary artifact

An artifact that can be directly correlated to a single exemplar system. A typical example would be code, or a design or requirements document that specifically addresses a given system. This would be contrasted to a secondary artifact like a survey article discussing several systems or talking about general functionality in an application area. See also secondary artifact.

process asset

Encoding, formalization, or automation of a work process in the domain, typically used as a tool, decision aid, etc., rather than a reusable component to be directly incorporated into system artifacts. Potential process assets mainly emerge in the *Interpret Domain Model* task within the *Refine Domain Model* sub-phase of *Model Domain*. In the *Select Features and Customers* task the surrounding context and rationale for these process assets is investigated. Often, the result can be discovery of potential types of assets never envisioned at the start of the domain engineering project. See also asset.

project

In the ODM context, a domain engineering project. See domain engineering project.

PROJECT CHARTER

A data item which provides bounding criteria for the ORGANIZATION CONTEXT of the project, including whether the initiative is part of a more comprehensive reuse program. The charter may be a formal document or may be implicit, based on the informal contextual knowledge of project members. The PROJECT CHARTER is a component of the ORGANIZATION CONTEXT data item.

Charters can take several forms, including the following:

- A mandate to perform domain engineering within a particular business area, selecting the domain and technology as appropriate;

- A mandate to validate or demonstrate a particular reuse technology;

- A mandate to achieve certain business objectives (time to market, productivity improvement) without specifically being directed to take a domain engineering approach.

PROJECT CONSTRAINTS

A data item consisting of restrictions imposed by organization and market conditions beyond the project scope. Constraints may include implicit expectations about project results, perceived risks, and other factors that could compromise the success of technically sound results.

Constraints may include the following:

- Technology constraints

- Constraints on domain selection

- Available PROJECT RESOURCES, as well as constraints on resources that can be tapped. For example, there may be a pragmatic constraint that the domain engineering effort cannot have any impact on the schedule or resources available for an application engineering project of high priority. Such a constraint can significantly shape viable objectives for the project.

- Restrictions on the scope of organization boundaries that the project can cross. For example, a domain engineering project in a software maintenance organization may have restricted access to organizations that developed application systems in the domain.

project member

A member of the domain engineering project team. A domain engineer.

PROJECT OBJECTIVES

A list of objectives for the domain engineering project mapped to stakeholder interests and the PROJECT CHARTER. Each objective is qualified as to whether it is an incentive or disincentive for each stakeholder based on each of the stakeholder's interests. Exhibit C-3, PROJECT OBJECTIVES/ STAKEHOLDERS INTERESTS MATRIX, contains a worksheet template that can be used to explore the impact of each objective on each stakeholder. Exhibit C-4, PROJECT STAKEHOLDERS/OBJEC-TIVES SUMMARY MATRIX, shows a worksheet template that can be used to map objectives against each stakeholder interest as a cross-check to show the level of commitment to each objective. PROJECT OBJECTIVES are documented during the *Identify Candidate Objectives* task and modified during the *Select Stakeholders and Objectives* task in the *Plan Domain* phase. Specific project objectives related to the domain of focus are added in the *Select Domain of Focus* task in the *Plan Domain* phase.

PROJECT OBJECTIVES/CRITERIA MATRIX

A worksheet which that can be used to map DOMAIN SELECTION CRITERIA to PROJECT OBJEC-TIVES. A template of this workproduct is included in Exhibit C-6. The PROJECT OBJECTIVES/CRI-TERIA MATRIX worksheet can be used in developing the DOMAIN SELECTION CRITERIA workproduct, during the *Define Selection Criteria* task in the *Plan Domain* phase.

PROJECT OBJECTIVES/STAKEHOLDER INTERESTS MATRIX

A worksheet which can be used to explore the impact of each objective on each stakeholder. A template of this worksheet is illustrated in Exhibit C-4. The PROJECT OBJECTIVES/STAKEHOLDER INTERESTS MATRIX worksheet can be used in developing the PROJECT OBJECTIVES workproduct, during the *Identify Candidate Objectives* task in the *Plan Domain* phase.

project stakeholder

A stakeholder with interests in the intended results of the project as defined by the PROJECT OBJECTIVES.

PROJECT STAKEHOLDER MODEL

The PROJECT STAKEHOLDER MODEL contains a list of the stakeholders whose interests will be sat-isfied by the selected PROJECT OBJECTIVES. The PROJECT STAKEHOLDER MODEL includes stake-holders from the STAKEHOLDER DOSSIER who are still of interest to the project and any new stakeholders that emerge as a result of selected objectives. Any relevant information from the dos-sier, such as descriptions of roles, relationships, interests, enablers and barriers, and assessments of attitudes towards reuse and reuse practices, can be referenced as needed in the model. If the roles of any stakeholders will change if the domain engineering project is successful, these role changes should also be described in the PROJECT STAKEHOLDER MODEL. For each stakeholder, the model should specify the following:

- Intended role on the project, including relations with other stakeholders;

- The anticipated benefit of the project to the stakeholder (their payoff);

- Their responsibilities (which can include support within management from sponsors, com-mitted resources, and direct participation);

449

PROJECT STAKEHOLDERS INTERESTS TABLE

A worksheet which can be used for identifying stakeholder interests, and enablers and barriers to stakeholder buy-in based on each of these interests. A template of this worksheet is include in Exhibit C-2. The PROJECT STAKEHOLDERS INTERESTS TABLE worksheet can be used in developing the STAKEHOLDER DOSSIER workproduct, during the *Determine Candidate Stakeholders* task in the *Plan Domain* phase.

PROJECT STAKEHOLDERS/OBJECTIVES SUMMARY MATRIX

A worksheet that can be used to map objectives against each stakeholder interest as a cross-check to show the level of commitment to each objective. A template of this workproduct is include in Exhibit C-4. The PROJECT STAKEHOLDERS/OBJECTIVES SUMMARY MATRIX worksheet can be used in developing the PROJECT OBJECTIVES workproduct, during the *Select Stakeholders and Objectives* task in the *Plan Domain* phase.

PROJECT STAKEHOLDERS/ROLES MATRIX

A worksheet which can be used check the list of stakeholders for completeness with respect to the fundamental domain engineering roles. A template of this workproduct is include in Exhibit C-2. The PROJECT STAKEHOLDERS/ROLES MATRIX worksheet can be used in developing the STAKEHOLDER DOSSIER workproduct, during the *Determine Candidate Stakeholders* task in the *Plan Domain* phase.

project team

The team performing the domain engineering project. This includes the domain planners, domain investigators, domain modelers, asset base engineers, and optionally certain domain informants. Sponsors, managers, and asset base customers are usually considered project stakeholders but not members of the project team.

related domain

A domain related to the *domain of focus* according to one of the domain relation types that form part of the DOMAIN CLASSIFICATION, DOMAIN INTERACTIONS or DOMAIN HISTORY components of the DOMAIN DEFINITION.

RELATED DOMAINS/DEFINING RULES MATRIX

A worksheet which can be used to characterize and classify related domains in terms of the DOMAIN DEFINING RULES. A template of this worksheet is illustrated in Exhibit C-9. The RELATED DOMAINS/DEFINING RULES MATRIX worksheet can be used in developing the Domain Classification workproduct, during the *Situate Domain* task in the *Plan Domain* phase.

representative system

An exemplar system which has been selected for intense analysis as a primary source of information during the *Describe Domain* sub-phase.

REPRESENTATIVE SYSTEM ARTIFACTS

A data item consisting of artifacts selected from representative system as sources of information for domain modeling. REPRESENTATIVE SYSTEM ARTIFACTS are a component of the EXEMPLAR SYSTEM ARTIFACTS data item.

representative system setting

For software domains, one of the settings associated with a representative system that has been chosen as a setting from which data will be gathered. (For non-software domains, the term would be simply a representative setting.)

REPRESENTATIVE SYSTEMS INFORMATION QUALITY MATRIX

A mapping of all systems in the REPRESENTATIVE SYSTEMS SELECTION against the DOMAIN SETTINGS. Exhibit C-11, REPRESENTATIVE SYSTEMS INFORMATION QUALITY MATRIX, shows a worksheet template that can be used for this activity. The mapping of representative systems against Domain Settings is used to determine system settings to examine. The REPRESENTATIVE SYSTEMS INFORMATION QUALITY MATRIX worksheet can be used to develop the DATA ACQUISITION PLAN workproduct, during the *Plan Data Acquisition* task in the *Model Domain* phase.

REPRESENTATIVE SYSTEMS SELECTION

A list of systems, annotated with rationale behind why they were considered to be a representative set for the purposes of domain engineering. The REPRESENTATIVE SYSTEMS SELECTION includes information about the set of system settings to be studied for each representative system. Each setting is related to one of the DOMAIN SETTINGS determined during the *Define Domain* sub-phase of the *Plan Domain* phase. Exhibit C-11, REPRESENTATIVE SYSTEMS INFORMATION QUALITY MATRIX, shows a worksheet template that can be used to create the mapping from representative systems to DOMAIN SETTINGS. The mapping is used to select system settings to examine. The REPRESENTATIVE SYSTEMS SELECTION workproduct is produced in the *Plan Data Acquisition* task in the *Model Domain* phase. See also system setting.

REUSE ATTRIBUTES

A data item consisting of general domain-independent attributes of promising domains for reuse.

reuse management

The management aspects of reuse addressing the establishment and continual improvement of reuse-oriented activities within an organization by emphasizing learning as an institutional mechanism for change. Learning in this context means actively evaluating and reflecting on behavior to effect positive change.

reuse program

The set of activities encompassed by (and including) a particular instance of reuse management.

reverse engineering

The process of analyzing a computer system's software to identify components and their interrelationships.

role

A position of responsibility to be occupied when performing work. People and tools hold roles in settings. See also setting.

scope of applicability

The range of application contexts for which an ASSET has been designed. Sometimes termed the intended scope of applicability, application scope, scope of application, market profile.

secondary artifact

An artifact without a clear correlation to a single exemplar system that serves as a source of domain information. See also primary artifact.

separate selectability

The ability of a particular functional subset or subsystem of an overall system to be extracted as a stand-alone system or component. An architecture which allows for various subsystems to be extracted in this way is a **separately selectable architecture.**

setting

The most general term for an environment where people interact with each other and perform processes. Settings imply a certain stability in that the same people work together on a routine basis. Synonymous terms used interchangeably are work setting and work practice setting. For the purposes of domain engineering in software-intensive domains, we are usually interested in system settings, settings where software systems are being developed or used or are otherwise involved in the flow of activities.

setting characteristic

An attribute of a setting. In the ODM context, setting characteristics are differentiating attributes of a particular kind of setting as defined in the DOMAIN SETTINGS. For example, a development setting might be a particular setting defined in the DOMAIN SETTINGS. Given the domain of focus, we might want to know what kind of operating system (OS) platform is being used in the development setting. The OS is not part of the domain focus; but knowing the OS used in a given setting helps determine what domain-specific features might be of interest to developers in that setting. In this example, "OS platform" would be a setting characteristic, and different specific system settings would have specific values associated with that setting.

situated

Used in two senses. The term implies being oriented in an environment where locality and the web of relationships have semantic meaning. The domain of focus is situated with respect to related models in the *Situate Domain* task. We also speak of a practitioner or an artifact being situated in a given setting. Often this usage implies that we, as investigators, have learned enough about the person or artifact that we are able to place them in the setting in order to correctly interpreting their behavior. See also context.

specialization

A specialization of a given entity has *additional* features and usually addresses a *narrower* scope of applicability. See also specialization subdomain.

specialization sub-domain

A domain D1 is said to *specialize* a domain D2 if:

- The exemplar set, or ORGANIZATION CONTEXT, of D1 is a subset of the exemplar set of D2; or

- The set of defining features for D1 is a *superset* of the defining features for D2 (i.e., a more tightly constrained definition).

This is an easy point of confusion. Consider the generality versus optimization trade-off: the narrower the ORGANIZATION CONTEXT for which a system is built, the more functionality can be

included in the system, e.g., it becomes more special-purpose. So as the context narrows, the set of features expands.

sponsor

A stakeholder who funds or otherwise initiates a domain engineering project, and typically writes or controls the PROJECT CHARTER.

stakeholder

A person, group or organizational entity that has interests and objectives relative to a system, project, or domain. Stakeholders have diverse viewpoints, experience, and terminology. Stakeholders include organization stakeholders, project stakeholders, and domain stakeholders. In a domain engineering project, these stakeholders can hold the role of a domain engineer, informant, customer, or practitioner.

stakeholder community

See stakeholder context.

stakeholder context

The set of stakeholders that form the ORGANIZATION CONTEXT the domain engineering project, together with their interests and relations. Depending on the nature of the domain engineering initiative and funding for the project, this context can be:

- a single organization or a division of a larger organization, such as a corporate or university research and development facility, a product-line division, or a system development group,

- multiple organizations, as typified by standards organizations and consortia with common interests in particular functional areas,

- a marketplace of varied competitor, partner and customer organizations, or

- a technical community.

Synonym for ORGANIZATION CONTEXT. This context may informally be referred to as "the organization" for convenience.

STAKEHOLDER DOSSIER

The STAKEHOLDER DOSSIER contains a list of candidate project stakeholders, partitioned into key and other stakeholders. The dossier also contains information about stakeholder roles, relationships, interests, enablers and barriers, and assessments of attitudes towards reuse and reuse practices. Exhibit C-1, PROJECT STAKEHOLDERS/ROLES MATRIX, contains a worksheet template that can be used check the list of stakeholders for completeness with respect to the fundamental domain engineering roles. Exhibit C-2, PROJECT STAKEHOLDERS INTERESTS TABLE, contains a worksheet template that can be used for identifying stakeholder interests, and enablers and barriers to stakeholder buy-in based on each of these interests. The STAKEHOLDER DOSSIER is used in coordination with candidate objectives to determine the final PROJECT OBJECTIVES in the *Select Stakeholders and Objectives* task. The STAKEHOLDER DOSSIER workproduct is produced in the *Determine Candidate Stakeholders* task of the *Plan Domain* phase.

stakeholder interest

A perceived benefit, risk, or other strategic motivator to a stakeholder with respect to the domain engineering project.

STAKEHOLDER KNOWLEDGE

A data item used to elicit terms used within the stakeholder community to discover PROJECT OBJECTIVES and DOMAINS OF INTEREST. STAKEHOLDER KNOWLEDGE is a component of the ORGANIZATION INFORMATION data item. Includes knowledge acquired from:

* literature which discusses or classifies application systems that may be of interest to candidate stakeholders, e.g., survey articles or tutorials by acknowledged experts; or

* direct interviews with informants that probe what domains are considered significant within the ORGANIZATION CONTEXT.

More generally, **stakeholder knowledge** is the knowledge stakeholders have, elicited via interviews, project participation, etc. Generally, when stakeholder knowledge is obtained it has a strategic role in the process; when informant knowledge is obtained it has more of a descriptive or informational role in the process. Stakeholder decisions are not knowledge; hence, processes intended to obtain commitments from stakeholders will not necessarily show STAKEHOLDER KNOWLEDGE as an input.

Often the term implies knowledge needed to assist in **context recovery**. For example, discussions with stakeholders may elicit the term "fault-tolerant mission-critical embedded systems." While couched in terms of a general class, there is probably a specific set of such systems of direct interest to the organization: e.g., "fault-tolerant mission-critical embedded systems maintained by Division X, but not the ones that run on the old XJK-77 systems..." This STAKEHOLDER KNOWLEDGE is usually common knowledge and therefore taken for granted, but may be important to document explicitly.

starter model

A model reused from non domain-specific infrastructure, previous domain engineering efforts that provides a starting point for descriptive modeling.

sub-domain

See component sub-domain and specialization sub-domain. The term "sub-domain" by itself is ambiguous in the ODM context and its use is generally avoided. When it occurs it is informal for component sub-domain or means both types of sub-domain.

sub-phase

Each of the three phases of domain engineering are further decomposed into sub-phases.

subsystem

A component of a system; usually but not necessarily encapsulated, i.e., appearing as a distinct component in the system architecture.

supporting methods

Methods that do not address domain engineering concerns but are needed to carry out domain engineering. These methods should be chosen by each organization based on their constraints, preferences, and domain. Supporting methods include system modeling techniques and taxonomic modeling techniques. A description of supporting methods that must be chosen for domain engineering appears in Section 8.0.

synonym

One of two or more words that have the same or nearly the same meaning. Used as a defined relation between terms in the DOMAIN LEXICON.

system

An application designed for a single usage setting. There is no intent to imply a systems (i.e., hardware, software, practitioners) versus software-only distinction. A system is developed and used by a set of practitioners performing activities, using tools and technology, within a set of system settings.

system architecture

The high level design of a software system. An architecture is defined in terms of the following general constructs:

- a set of software system elements, which may include both processing and data elements
- interfaces for each element
- a set of element-to-element connections, collectively forming interconnection topologies
- the semantics of each connection
    - the meaning of static connections (e.g., between data elements)
    - protocols describing information transfer across dynamic connections, in terms of element interfaces (general classes of protocols include procedure call, pipe, message passing, etc.

system artifact

An artifact used in the development or execution of a system (e.g., a requirements document, design document, executable module, system start-up file). See also artifact.

system engineering

Implementing a new system for a single application context. What constitutes a single application context will vary from organization to organization. Contrasts with variability engineering, which involves a specified set of multiple application contexts, and domain engineering, which involves strategic selection of a set of multiple contexts.

SYSTEM INFORMATION

SYSTEM INFORMATION is a data item component of the ORGANIZATION INFORMATION data item. This is specifically information about **systems of interest**, including documentation from any phase of the life cycle, product plans, market studies, usage reports, etc.

system life cycle

All of the settings for a given exemplar taken as a whole.

system practitioner

A person who plays one or more practitioner roles within one or more system settings. See also practitioner.

system reengineering

Restructuring and/or reimplementing some or all of an existing legacy system.

system setting

A **work practice setting** that involves the development, use, maintenance etc. of a [software] system. See also setting.

system of interest

A system which is of interest to one or more of the stakeholder organizations that form the ORGANIZATION CONTEXT for a domain engineering project. This could include systems developed by, maintained by, or used by the organization; or systems that have competitive impact in a product marketplace. After domain selection, subset of the systems of interest are identified as CANDIDATE EXEMPLAR SYSTEMS.

target system

A system into which assets are to be incorporated or the development process of which will in some way be encapsulate as a reusable asset. Target systems can be considered the dual of exemplar systems: i.e., exemplar systems yield artifacts, studied in descriptive modeling, transformed via prescriptive asset base engineering into assets, incorporated into target systems. An exemplar system studied in the *Plan Domain* or *Model Domain* phases may but need not become a target system in the *Engineer Asset Base* phase; conversely, target systems may but need not have been exemplar systems. As with exemplar systems, target systems can be legacy systems or anticipated new systems.

task

One of the 27 "leaf nodes" of the process tree that represent a primary task in the ODM life cycle.

taxonomic modeling

A supporting methods described in Section 8.0. Taxonomic modeling is used to describe the commonality and variability in a set of exemplars. It may involve hierarchical classification schemes, more formal semantic networks, simpler faceted schemes, or other representations. It contrasts, in the ODM context, with system modeling, which focuses on structural and behavioral descriptions of individual systems. In practice some integration between the two modeling representations must be supported. Variability modeling is a specialization of taxonomic modeling used in the ODM context to specifically addresses variability within software systems. See also variability modeling.

team modeling

One of the optional layers described for ODM. Many aspects of the ODM method require different sorts of team interaction skills than those customary for groups of software developers. The modeling process works by continual interaction between centralized and distributed modeling tasks. Several groups may build related models relatively independently, then meet to integrate the models and resolve the inconsistencies. Model development is partitioned into reasonably separate tasks, which can be performed iteratively, in parallel or even in round-robin fashion, with circulation of personnel between different modeling and data-gathering tasks. This separate team modeling strategy is necessary, even when models produced are conceptually linked together and could in principle be thought of as one large model. This is considered an optional layer because a team modeling approach can be used throughout the ODM life cycle.

template

See worksheet template.

term

An entry in the DOMAIN TERMS or DOMAIN LEXICON.

term cluster

A group of terms interpreted by the modeler as synonymous, and gathered together in a term cluster, with a a standard canonic term as a representative. The term cluster may include acronyms or contextual abbreviations of compound terms as well as true synonyms. For example, "domain exemplar" and "exemplar" could be considered a cluster, where "domain exemplar" more fully qualifies the term "exemplar". Within a given domain setting, a given term may be assigned a semantics that overlaps with the most commonly intended compound term. For example, it is not necessary in the ODM context to always use domain exemplars as a term, since *domain* exemplar is understood whenever the term exemplar is used. Use of the term exemplar without a meaning of domain exemplar would need to be made explicit. Other syntactic variations can be encompassed as part of the term cluster: e.g., case, voice (e.g., active-passive) and plurality variants, etc. For example: "variable" and "variability," "bounds" and "bounded by," "concept model" and "concept models." See also canonic term.

TEST AND VALIDATION PLAN

A plan for how individual ASSETS, the ASSET BASE INFRASTRUCTURE, and the ASSET BASE as a whole will be tested and validated. This plan is based on the types of assets to be implemented, the implementation techniques selected, and the phased development plan. It may involve planning for the development and use of asset test cases (e.g., to test if an asset supports the desired range of variability) and of *test* (and *test plan*) *assets* that can be instantiated to support testing of the system artifacts derived from other assets (note that the detailed implementation of such assets should be covered in the ASSET IMPLEMENTATION PLAN). The TEST AND VALIDATION PLAN is created in the *Plan Asset Base Implementation* task in the *Engineer Asset Base* phase.

usage setting

A setting where a software system executes its functions, usually in interaction with human beings.

utility

The level of need or interest in a feature or feature set among the candidate customers. See also feasibility.

value chain

Characterizes producer-consumer relationships between a series of stakeholder settings, beginning with the asset developer and ending with the application end-user.

variability

The ways in which two concepts or entities differ. Used in opposition with commonality. See also commonality.

variability engineering

Application and extension of systems engineering techniques to multiple-system settings, such as product-line planning and engineering. Encompasses variability modeling, as well as various design and implementation techniques to cost-effectively support variability, via flexible architectures, generative technologies, etc. Variability engineering is a sub-problem of domain engineering; the term implies that there is an established set of customer settings that impose definite, although variable, requirements on the engineering task.

For example, a requirements specification may specify that a given system must be portable across three specific hardware platforms. In a full-scale domain engineering problem as defined within ODM, the selection of the specific platforms to support (and how many to support) would be part of the engineering task. See also variability modeling, variability reengineering.

variability modeling

The problem of how to represent variability both within and across systems. It includes the problem of distinguishing the binding sites of various functions within both development and usage settings.

For example, some object-oriented programming languages can express run-time polymorphism in data structures and procedures. Languages like Ada83 use generics that are processed largely at compile-time (simplified for the sake of example); code generators can be written to produce optimized procedures in languages that support no polymorphism at all. Variability modeling would address the problem of how to represent these (and possibly other) design options in a single, integrated design space. Variability engineering would address the problem of deciding which option to use. See also variability engineering.

variant

Two exemplar artifacts are variants if they are classified as instances of a common category but have distinguishing features. The term can also be applied to features themselves, although this implies particular capabilities in the taxonomic modeling representations used. See also architectural variant.

vertical domain

Domains that include entire systems in their scope. These systems may be closely related (e.g., a set of system versions or a product line) or loosely related (a family of systems). In ODM the terms *vertical* and *horizontal* are not absolute qualities of general functional areas, but describe the span of a given domain's coverage relative to a particular set of systems. For example, within a family of large-scale systems, database management may be considered a horizontal domain. In the Database Management Systems provider marketplace, database management could be considered a vertical domain. See also horizontal domain.

work practice setting

See system setting.

work setting

See system setting.

workproduct

A textual or graphical document, or a model, produced during a domain engineering project task that exists in persistent form. In this document, this term is used *only* for ODM workproducts. All

documents, system models, code modules, etc. from representative systems are considered arti-facts from the standpoint of ODM, even though within the context of the application projects themselves the same documents would be considered workproducts. By analogy, if researchers were to collect samples of ODM workproducts from multiple domain engineering projects and compare them for commonality and variability, the documents would be playing the role of arti-facts. See also composite workproduct, component workproduct.

worksheet

Suggest optional ways of creating a part of a workproduct or carrying out a given activity. Some-what analogous to the "worksheets" used in tax preparation, these are ways to help you get to the main results of the task. Worksheets are rarely passed as data across tasks, and are not shown as separate data flows on the $IDEF_0$ diagrams. They are generally described within the Activities descriptions rather than in the Workproducts description. Examples and templates for some (not all) worksheets are provided, respectively, at the end of the appropriate task section and in Appendix C.2.

worksheet template

The worksheet templates are intended to suggest how some of the content of specific workprod-ucts can be organized and formatted. In general, they are not usable as-is in any specific context because, for example, the numbers of rows and columns (and associated labels) in most of these tabular worksheets will change depending on the domain, the number of exemplars, the number of stakeholders, etc. Thus, the templates will need to be adapted for use on specific projects. The worksheet templates are shown in Section C.2.

Appendix C: ODM Workproducts

This appendix contains a quick-reference catalog of the ODM workproducts, data items, and worksheets in Section C.1, followed by a set of worksheet templates in Section C.2.

C.1 Workproduct Catalog

This section provides two itemized catalogs of the ODM workproducts, data items, and worksheets. These are intended to offer a quick reference to these ODM items. They can be used as a kind of index into the more detailed descriptions of these items in the ODM Lexicon in Appendix B, as well as the process description sections in which the ODM items are created and used.

The two catalogs are organized in different ways. The first places them in the following categories:

- *Composite Workproduct* – Workproduct produced by a phase or sub-phase, consisting of one or more other workproducts produced by lower-level processes

- *Workproduct* – Products produced by tasks

- *Data Item* – Data originating outside the ODM process model

- *Worksheet* – Formatted data that can contribute to or be part of a workproduct

and lists them alphabetically in each category.

The second catalog lists the ODM tasks in the order they are presented in Part II and, under each task, lists the ODM workproducts produced by that task and the worksheet templates that can be applied in producing the workproducts.

C.1.1 Workproduct Catalog Organized by Category

Composite Workproducts (3)
> ASSET BASE
> DESCRIPTIVE MODELS
> DOMAIN DEFINITION

Workproducts (43)
> ASSET BASE ARCHITECTURE
> ASSET BASE DOSSIER
> ASSET BASE INFRASTRUCTURE
> ASSET BASE MODEL
> ASSET IMPLEMENTATION PLAN
> ASSET SPECIFICATIONS
> ASSETS
> ASSETS OF INTEREST
> CANDIDATE EXEMPLAR SYSTEMS
> CONCEPT MODELS
> CONCEPTS OF INTEREST
> CUSTOMER DOSSIER
> DATA ACQUISITION PLAN

461

DOMAIN CLASSIFICATION
DOMAIN DATA
DOMAIN DEFINING RULES
DOMAIN DESCRIPTION
DOMAIN DOSSIER
DOMAIN HISTORY
DOMAIN INTERACTIONS
DOMAIN LEXICON
DOMAIN MODEL
DOMAIN SELECTION
DOMAIN SELECTION CRITERIA
DOMAIN SETTINGS
DOMAIN STAKEHOLDER MODEL
DOMAIN TERMS
DOMAINS OF INTEREST
EXEMPLAR SYSTEMS SELECTION
EXTERNAL ARCHITECTURE CONSTRAINTS
FEATURE MODELS
FEATURES OF INTEREST
INFRASTRUCTURE IMPLEMENTATION PLAN
INTEGRATED DOMAIN MODEL
INTERNAL ARCHITECTURE CONSTRAINTS
INTERPRETED DOMAIN MODEL
NEW DOMAIN TERMS
NEW SYSTEM ARTIFACTS
PROJECT OBJECTIVES
PROJECT STAKEHOLDER MODEL
REPRESENTATIVE SYSTEMS SELECTION
STAKEHOLDER DOSSIER
TEST AND VALIDATION PLAN

Data Items (15)

ASSET BASE CUSTOMER INFORMATION
CANDIDATE CUSTOMER INFORMATION
DOMAIN ARTIFACTS
DOMAIN INFORMANT KNOWLEDGE
DOMAIN INFORMATION
DOMAIN STAKEHOLDER KNOWLEDGE
EXEMPLAR SYSTEM ARTIFACTS
ORGANIZATION CONTEXT
ORGANIZATION INFORMATION
PROJECT CHARTER
PROJECT CONSTRAINTS
REPRESENTATIVE SYSTEM ARTIFACTS
REUSE ATTRIBUTES
STAKEHOLDER KNOWLEDGE
SYSTEM INFORMATION

Worksheets (17)

BUSINESS OPPORTUNITIES TABLE
    Can be used to produce the CUSTOMER DOSSIER workproduct.

COMPONENT CONSTRAINTS TABLE
    Can be used to produce the INTERNAL ARCHITECTURE CONSTRAINTS workproduct.

CONCEPTUAL/HISTORICAL RELATIONS MATRIX
    Can be used to validate the DOMAIN CLASSIFICATION and DOMAIN HISTORY workproducts.

CUSTOMER/EXTERNAL INTERFACE MATRIX
    Can be used to produce the EXTERNAL ARCHITECTURE CONSTRAINTS workproduct.

DOMAINS/CRITERIA MATRIX
    Can be used to produce the DOMAIN SELECTION workproduct.

DOMAINS/SYSTEMS MATRIX
    Can be used to produce the DOMAINS OF INTEREST workproduct.

EXEMPLARS/DEFINING RULES MATRIX
    Can be used to validate the DOMAIN DEFINING RULES and EXEMPLAR SYSTEMS SELECTION workproducts.

FEATURE/EXTERNAL INTERFACE MATRIX
    Can be used to produce the EXTERNAL ARCHITECTURE CONSTRAINTS workproduct.

FEATURE/CUSTOMER MATRIX
    Can be used to produce the CUSTOMER DOSSIER, ASSET BASE DOSSIER, and ASSET BASE MODEL workproducts.

LAYERING CONSTRAINTS TABLE
    Can be used to produce the INTERNAL ARCHITECTURE CONSTRAINTS workproduct.

PROJECT OBJECTIVES/CRITERIA MATRIX
    Can be used to produce the DOMAIN SELECTION CRITERIA workproduct.

PROJECT OBJECTIVES/STAKEHOLDERS INTERESTS MATRIX
    Can be used to produce the PROJECT OBJECTIVES workproduct.

PROJECT STAKEHOLDERS INTERESTS TABLE
    Can be used to produce the STAKEHOLDER DOSSIER workproduct.

PROJECT STAKEHOLDERS/OBJECTIVES SUMMARY MATRIX
    Can be used to produce the PROJECT OBJECTIVES workproduct.

PROJECT STAKEHOLDERS/ROLES MATRIX
    Can be used to produce the STAKEHOLDER DOSSIER workproduct.

RELATED DOMAINS/DEFINING RULES MATRIX
    Can be used to produce the DOMAIN CLASSIFICATION workproduct.

REPRESENTATIVE SYSTEMS INFORMATION QUALITY MATRIX
    Can be used to produce the REPRESENTATIVE SYSTEMS SELECTION workproduct.

C.1.2 Workproduct Catalog Organized by Tasks

Determine Candidate Stakeholders — Section 5.1.1

- Workproducts:
  STAKEHOLDER DOSSIER

- Worksheets:
  PROJECT STAKEHOLDERS INTERESTS TABLE
  PROJECT STAKEHOLDERS/ROLES MATRIX

Identify Candidate Objectives — Section 5.1.2

- Workproducts:
  PROJECT OBJECTIVES

- Worksheets:
  PROJECT OBJECTIVES/STAKEHOLDERS INTERESTS MATRIX

Select Stakeholders and Objectives — Section 5.1.3

- Workproducts:
  PROJECT OBJECTIVES
  PROJECT STAKEHOLDER MODEL

- Worksheets:
  PROJECT STAKEHOLDERS/OBJECTIVES SUMMARY MATRIX

Characterize Domains of Interest — Section 5.2.1

- Workproducts:
  DOMAINS OF INTEREST

- Worksheets:
  DOMAINS/SYSTEMS MATRIX

Define Selection Criteria — Section 5.2.2

- Workproducts:
  DOMAIN SELECTION CRITERIA

- Worksheets:
  PROJECT OBJECTIVES/CRITERIA MATRIX

Select Domain of Focus — Section 5.2.3

- Workproducts:
  CANDIDATE EXEMPLAR SYSTEMS
  DOMAIN SELECTION
  DOMAIN STAKEHOLDER MODEL
  PROJECT OBJECTIVES

- Worksheets:
  DOMAINS/CRITERIA MATRIX

Bound Domain — Section 5.3.1

- Workproducts:
  DOMAIN DESCRIPTION
  EXEMPLAR SYSTEMS SELECTION
  DOMAIN DEFINING RULES

- Worksheets:
  EXEMPLARS/DEFINING RULES MATRIX

Focus Domain — Section 5.3.2

- Workproducts:
  DOMAIN SETTINGS
  DOMAIN DEFINING RULES

- Worksheets:
  None

Situate Domain — Section 5.3.3

- Workproducts:
  DOMAIN CLASSIFICATION
  DOMAIN HISTORY
  DOMAIN INTERACTIONS

- Worksheets:
  RELATED DOMAINS/DEFINING RULES MATRIX
  CONCEPTUAL/HISTORICAL RELATIONS MATRIX

Plan Data Acquisition — Section 6.1.1

- Workproducts:
  DATA ACQUISITION PLAN
  REPRESENTATIVE SYSTEMS SELECTION

- Worksheets:
  REPRESENTATIVE SYSTEMS INFORMATION QUALITY MATRIX

Elicit Data — Section 6.1.2

- Workproducts:
  DOMAIN DATA
  NEW SYSTEM ARTIFACTS

- Worksheets:
  None

Integrate Data — Section 6.1.3

- Workproducts:
  DOMAIN TERMS
  DOMAIN DOSSIER
  FEATURES OF INTEREST

- Worksheets:
  None

Develop Lexicon — Section 6.2.1

- Workproducts:
  DOMAIN LEXICON

- Worksheets:
  None

Model Concepts — Section 6.2.2

- Workproducts:
  CONCEPT MODELS
  NEW DOMAIN TERMS
  CONCEPTS OF INTEREST

- Worksheets:
  None

Model Features — Section 6.2.3

- Workproducts:
  FEATURE MODELS
  NEW DOMAIN TERMS

- Worksheets:
  None

Integrate Descriptive Models — Section 6.3.1

- Workproducts:
  INTEGRATED DOMAIN MODEL

- Worksheets:
  None

Interpret Domain Model — Section 6.3.2

- Workproducts:
  INTERPRETED DOMAIN MODEL

- Worksheets:
  None

Resolve Domain Model — Section 6.3.3

- Workproducts:
  DOMAIN MODEL

- Worksheets:
  None

Correlate Features and Customers — Section 7.1.1

- Workproducts:
  CUSTOMER DOSSIER

- Worksheets:
  BUSINESS OPPORTUNITIES TABLE
  FEATURE/CUSTOMER MATRIX

Prioritize Features and Customers — Section 7.1.2

- Workproducts:

ASSET BASE DOSSIER

- Worksheets:
  FEATURE/CUSTOMER MATRIX

Select Features and Customers — Section 7.1.3

- Workproducts:
  ASSET BASE MODEL
  ASSETS OF INTEREST

- Worksheets:
  FEATURE/CUSTOMER MATRIX

Determine External Architecture Constraints — Section 7.2.1

- Workproducts:
  EXTERNAL ARCHITECTURE CONSTRAINTS

- Worksheets:
  CUSTOMER/EXTERNAL INTERFACE MATRIX
  FEATURE/EXTERNAL INTERFACE MATRIX

Determine Internal Architecture Constraints — Section 7.2.2

- Workproducts:
  INTERNAL ARCHITECTURE CONSTRAINTS

- Worksheets:
  COMPONENT CONSTRAINTS MATRIX
  LAYERING CONSTRAINTS MATRIX

Define Asset Base Architecture — Section 7.2.3

- Workproducts:
  ASSET BASE ARCHITECTURE
  ASSET SPECIFICATIONS

- Worksheets:
  None

Plan Asset Base Implementation — Section 7.3.1

- Workproducts:
  ASSET IMPLEMENTATION PLAN
  INFRASTRUCTURE IMPLEMENTATION PLAN
  TEST AND VALIDATION PLAN

- Worksheets:
  None

Implement Assets — Section 7.3.2

- Workproducts:
  ASSETS

- Worksheets:
    None

Implement Infrastructure — Section 7.3.3

- Workproducts:
    ASSET BASE INFRASTRUCTURE

- Worksheets:
    None

C.2 Worksheet Templates

The worksheet templates below are intended to suggest how some of the content of specific work-products can be organized and formatted. In general, they are not usable as-is in any specific context because, for example, the numbers of rows and columns (and associated labels) in most of these tabular worksheets will change depending on the domain, the number of exemplars, the number of stakeholders, etc. Thus, the templates will need to be adapted for use on specific projects.

Each template is preceded by a line identifying the ODM workproduct (or workproducts) to which the worksheet is *primarily* applicable, the corresponding ODM process (or processes) in which the workproduct(s) is/are produced, and the guidebook section number(s) in which the process(es) is/are described. Note that there may be cases where one template, suggested as being primarily useful within a given process, could be easily adapted for use in other processes. An example of this "template reuse" can be seen in the similarity between the PROJECT OBJECTIVES/ STAKEHOLDERS INTEREST MATRIX and PROJECT STAKEHOLDERS/OBJECTIVES SUMMARY MATRIX templates. As another example, several of the templates suggested for use in *Plan Domain* may also be relevant to *Scope Asset Base*.

Use these templates creatively and not literally to produce a concrete set of workproducts for your project.

C.2.1 Project Stakeholders/Roles Matrix

Used in the *Determine Candidate Stakeholders* task in Section 5.1.1 to produce the STAKE-HOLDER DOSSIER workproduct.

Stakeholders		Roles			
		Role 1	Role 2	Role 3	...
Key Stakeholders	Stakeholder 1				
	Stakeholder 2				
	Stakeholder 3				
	...				
Other Stakeholders	Stakeholder 1				
	Stakeholder 2				
	Stakeholder 3				
	...				

Exhibit C-1. Project Stakeholders/Roles Matrix

Key:

Roles that are relevant for the project are selected before filling out this table. A description of how roles relate to stakeholders, and some suggestions for roles are given in the text.

Each cell is marked with an X if the stakeholder potentially plays the corresponding role in the Domain Engineering project.

C.2.2 Project Stakeholder Interests Table

Used in the *Determine Candidate Stakeholders* task in Section 5.1.1 to produce the STAKE-
HOLDER DOSSIER workproduct.

Stakeholders Interests			Enablers	Barriers
Key Stakeholders	Stakeholder 1	interest 1.1		
		interest 1.2		
		...		
	Stakeholder 2	interest 2.1		
		interest 2.2		
		...		
	...	...		
Other Stakeholders	Stakeholder 3	interest 3.1		
		interest 3.2		
		...		
	...	...		

Exhibit C-2. Project Stakeholders Interests Table

Key:

Stakeholder interests are general interests; a single stakeholder may have many interests that are
relevant to the project objectives.

For each of these interests, some of them have an aspect that would enable the project, ii.e., what
about the project could cause the stakeholder to buy-in on the basis of this interest?

Some of them would act as barriers to the project; what would turn the stakeholder off, with
respect to this interest?

It is not necessary to fill in every cell, but it is worth considering each of them carefully, before
leaving them blank.

C.2.3 Project Objectives/Stakeholders Interest Matrix

Used in the *Identify Candidate Objectives* task in Section 5.1.2 to produce the PROJECT OBJEC-TIVES workproduct.

Candidate Objectives	Stakeholder Interests								
	Stakeholder 1			Stakeholder 2			...		
	Interest 1	Interest 2	...	Interest 1	Interest 2	...			
Objective 1②									
Objective 2	①					①			
Objective 3									
Objective 4	①					①			
...									

Exhibit C-3. Project Objectives/Stakeholders Interests Matrix

Key:

① Rate objectives, according to stakeholder interests. If an objective meets a barrier, then the interest is a disincentive for the objective. If an objective matches an enabler, then it is an incentive. Rate from incentive to disincentive thus:

　!. - Great incentive (non-negotiable)
　+ - Incentive (nice to have)
　0 - Neutral
　- - Disincentive (conflicts)
　X - Great disincentive (non-negotiable)
　? - Unknown, uncertain

② Generating objectives is an open-ended job - rows may be added to the table at will. It is often useful to organize objectives in a hierarchical outline format, to discriminate sub-objectives.

C.2.4 Project Stakeholders/Objectives Summary Matrix

Used in the *Select Stakeholders and Objectives* task in Section 5.1.3 to produce the PROJECT OBJECTIVE workproduct.

Project Stakeholders (Selected & Prioritized)		Project Objectives (Selected & Prioritized)			
		Objective 1	Objective 2	Objective 3	...
Key Stakeholders	Key Stakeholder 1				
	Key Stakeholder 2				
	...				
Other Stakeholders	Other Stakeholder 1				
	Other Stakeholder 2				
	...				

Exhibit C-4. Project Stakeholders/Objectives Summary Matrix

Key:

+ Strong incentive

0 Neutral

- Strong disincentive

This is the final sorted, prioritized, summary of the stakeholders/objectives matrix. Cells are filled with a summary of the incentive for each remaining stakeholder, taking into account all his/her interests. Annotations can also be made in cells for particular risks, mitigating strategies, etc.

473

C.2.5 Domain /Systems Matrix

Used in the *Characterize Domains of Interest* task in Section 5.2.1 to produce the DOMAINS OF INTEREST workproduct.

Domains of Interest	Systems of Interest ①			
	S1	S2	...	Sn
D1 ②	③			
D2				
...				
Dn				

Exhibit C-5. Domains/Systems Matrix

Key:

① Input to the task, established by ORGANIZATION CONTEXT
② Domains of Interest
③ Cells coded with:
    V -Vertical domain with respect to this system
    E - Encapsulated horizontal domain
    D - Diffused
    blank- Not applicable

C.2.6 Project Objectives/Criteria Matrix

Used in the *Define Selection Criteria* task in Section 5.2.2 to produce the DOMAIN SELECTION CRITERIA workproduct.

Project Objectives	Domain Selection Criteria			
	Criterion 1	Criterion 2	Criterion 3	...
Objective 1	①			
Objective 2				
Objective 3			①	
...				

Exhibit C-6. Project Objectives/Criteria Matrix

Key:

① Cells are filled with one of the following:

+   Satisfying this criterion supports this objective

0   Satisfying this criterion is neutral with respect to this objective

X   Satisfying this criterion conflicts with this objective

Rationale for the choice can be included in short notes written in the cells, or as footnotes.

475

C.2.7 Domains/Criteria Matrix

Used in the *Select Domain of Focus* task in Section 5.2.3 to produce the DOMAIN SELECTION workproduct.

Candidate Domains of Focus	Domain Selection Criteria (prioritized)			
	Cr 1	Cr 2	...	Cr n
Candidate Domain 1				
Candidate Domain 2				
...				
Candidate Domain n				

Exhibit C-7. Domains/Criteria Matrix

Used in the final trade-off analysis for selecting the domain. Criteria can be ordered by priority, or can be weighted with summary totals calculated and compared for each candidate domain. Additional factors particular to individual domains can be annotated in an extra column.

Key:

① Cells are filled with one of the following:
+   The candidate domain satisfies this criterion
0   The candidate domain is neutral with respect to this criterion
X or -   The candidate domain conflicts with/does not satisfy this criterion

C.2.8 Exemplars/Defining Rules Matrix

Used in the *Bound Domain* task in Section 6.3.1 to validate the DOMAIN DEFINING RULES and EXEMPLAR SYSTEMS SELECTION workproducts.

Candidate Exemplar Systems		Defining Rules ②					
	Classify ④	Rule 1	Rule 2	...			Rule n
Candidate Exemplar 1 ①	C						
Candidate Exemplar 2	E						
...	...						
			③		③		
				③			
		③					
Candidate Exemplar n							

Exhibit C-8. Exemplars/Defining Rules Matrix

Key:

①   List candidate exemplar systems

②   Convert informal definition to defining rules

③   Select all rules for each candidate (e.g., yes/no)

④   Classify each candidate:
     E   - Exemplar
     B   - Borderline
     C   - Counter

Each counter-exemplar should *fail* at least one defining rule. Borderline exemplars match some rules and fail at least one.

Note: Rules can be rules of inclusion or exclusion; the worksheet will be easiest to use if all rules can be expressed in common form, or if inclusive and exclusive rules are in separate sections.

C.2.9 Related Domains/Defining Rules Matrix

Used in *Situate Domain* task in Section 6.3.3 to produce the DOMAIN CLASSIFICATION workproduct.

Related Domains	Defining Rules (refer to columns from Exhibit C-6)						Differentiating Features
Generalization Domain Name ⑤ _____ _____	✓	✓	✓	X		X ①	②
Domain Name ⑤ _____ _____							
Analogy Domain Name ⑤ _____ _____	✓	✓	✓	X	X	✓ ③	③
Domain Name ⑤ _____ _____							
Specialization Domain Name ⑤ _____ _____	✓	✓	✓	✓	✓	✓	④
Domain Name ⑤ _____ _____							

Exhibit C-9. Related Domains/Defining Rules Matrix

Note: Each cell of the Differentiating Features column should be annotated with a brief description of one or more *additional* rules or features that differentiate the analogy or specialization domain from the domain of focus.

Key:

① Generalization domains should have at least one unsatisfied domain defining rule

② Generalization should *not* have differentiating features (otherwise = analogy)

③ Analogy should have at least one unsatisfied defining rule, plus some differentiating feature(s)

④ Specialization should have *all* defining rules, plus some differentiating feature(s)

⑤ Borderline or counter exemplars re-classified in this related domain (from Exhibit C-8 rows)

478

C.2.10 Conceptual/Historical Relations Matrix

Used in the *Situate Domain* task in Section 5.3.3 to validate the DOMAIN CLASSIFICATION and DOMAIN HISTORY workproducts.

	Past Systems	Current Systems	Future Systems (Forecasting)
Generalization	Early limited function systems	General platforms	Modularization "Lite" configurations
Analogy	Manual precursors Leveraged reuse	Analogies used by: • Designers • Interfacers • User training	Dual use / spin-off
Specialization	Custom legacy systems	Tailoring to specific environments Values-added extensions	Layered products Customization

Exhibit C-10. Conceptual/Historical Relations Matrix

Purpose of this template: correlate conceptual and historical relations to the domain of focus, by tracking development of the domain over time. A downward slope (past generalization -> future specialization) shows a trend towards specialization of domain functionality. An upward slope (past specialization -> future generalization) shows a trend toward widened applicability through reducing specialized functions. This can help identify long-term evolutionary trends and forecast future directions.

Key: This template can be used in various ways:

- To map relationships between *candidate exemplars* in the domains fill the cells with names of border or counter exemplars (taken from rows of the worksheet shown in Exhibit C-8).

- To map relationships between *domains* fill the cells with names of related domains (taken from major rows of the worksheet shown in Exhibit C-9).

- For either exemplars or domains, the worksheet can also be used as a graph by placing entries in "scatter-plot" mode (positional relation to each other, dropping row and column headings).

C.2.11 Representative Systems Information Quality Matrix

Used in the *Plan Data Acquisition* task in Section 6.1.1 to produce the REPRESENTATIVE SYSTEMS SELECTION workproduct.

Candidate Representative Systems	Domain Settings			
	Development Setting	Other Setting	Other Setting	Usage Setting
Representative System 1	H/E			
Representative System 2				
...				
Representative System n				

Exhibit C-11. Representative Systems Information Quality Map

Key:

Each cell summarizes all system settings corresponding to the domain setting, and rates both:

- Quality of Info:
  - H - High
  - L - Low
- Ease of Accessibility:
  - D - Difficult
  - E - Easy

C.2.12 Business Opportunities Table

Used in the *Correlate Features and Customers* task in Section 7.1.1 to produce the CUSTOMER DOSSIER workproduct.

Feature Sets	Stakeholder Settings ②			
	Asset Producer	Setting X	Application Developer	End User
Scenario 1: • Feature Set 1.1 ① • ... • Feature Set 1.m ①	③ Scenario Notes ⑤	③ Profiles <x,y> ④	③ Profiles <x,y> ④	③ Profiles <x,y> ④
...				
Scenario n: • Feature Set n.1 ① • ... • Feature Set n.m ①	③ Scenario Notes ⑤	③ Profiles <x,y> ④	③ Profiles <x,y> ④	③ Profiles <x,y> ④

Exhibit C-12. Business Opportunities Table

Key:

① Feature sets from ASSET BASE MODEL that are relevant to the scenario

② Stakeholder settings - typically will include Asset Producer, Application Developer, and End User, as well as other relevant settings, denoted by "Setting X" above.

③ Specific stakeholder/setting identifiers

④ The potential market profile/setting characteristics from the ASSET BASE MODEL which match candidate customer settings

⑤ Notes describing the motivations, activities, interactions, etc., associated with the scenario

C.2.13 Feature/Customer Matrix

Used in: the *Correlate Features and Customers* task in Section 7.1.1 to produce the CUSTOMER DOSSIER workproduct; the *Prioritize Features and Customers* task in Section 7.1.2 to produce the ASSET BASE DOSSIER workproduct; and the *Select Features and Customers* task in Section 7.1.3 to product the ASSET BASE MODEL workproduct.

Candidate Feature Sets	Candidate Customer Settings			
	CS1 ③	CS2 ③	...	CSx ③
① F1 <Descriptor>	⑤	⑤	d)	
Profile: ② F1A _____	④	④		b)
② F1B _____	④			b)
....	a)	a)		a)
② F1x _____				b)
① F2 <Descriptor>	⑤ c)	⑤ c)	d)	c)
Profile: F2A _____	④			
F2B _____		④		
...				
F2x _____				
...				
Fx, etc.			d)	

Exhibit C-13. Feature/Customer Matrix

Key:

① Feature sets, tagged by short descriptors
② For each feature set, list of attributes of potential market profile
③ List of customer settings
④ In the cells:
  How close a match is this attribute to this setting?
  ! - Close match/direct hit   - - Poor match/risk
  + - Good match   X - Non-match
  N - Neutral   ? - Unknown/uncertain
  f Summarize match of feature set to each setting

Notes:

a) Each attribute can match with multiple customers/settings
b) Each customer/setting can match with multiple attributes
c) A feature set as a whole can match with multiple customers/settings
d) A customer/setting can match with multiple feature sets

C.2.14 Feature/External Interface Matrix

Used in the *Determine External Architecture Constraints* task in Section 7.2.1 to produce the
EXTERNAL ARCHITECTURE CONSTRAINTS workproduct.

Asset Base Feature	Settings and Interfaces								
	② Development Setting			X Setting			Usage Setting		
	Id1	Id2	...	Ix1	Ix2	...	Iu1	Iu2	...
Feature 1 ①	③								
Feature 2				N/A	N/A	N/A			
...									
Feature n									

Exhibit C-14. Feature/External Interface Matrix

Key:

① Individual features from ASSET BASE MODEL; in prioritized order
② Settings and interfaces from the ASSET BASE MODEL
③ Indicate allocation of features to interfaces
X Setting- Other possible settings

C.2.15 Customer/External Interface Matrix

Used in the *Determine External Architecture Constraints* task in Section 7.2.1 to produce the EXTERNAL ARCHITECTURE CONSTRAINTS workproduct.

Asset Base Customer	Settings and Interfaces								
	② Development Setting			X Setting			Usage Setting		
	Id1	Id2	...	Ix1	Ix2	...	Iu1	Iu2	...
Customer 1 ①	③								
Customer 2				N/A	N/A	N/A			
...									
Customer n									

Exhibit C-15. Customer/External Interface Matrix

Key:

① Customers from ASSET BASE MODEL; in prioritized order
② Settings and interfaces from the ASSET BASE MODEL
③ In each cell, key constraints on the interface for that customer
   N/A if interface and/or setting does not apply to that customer
X Setting- Other possible settings

C.2.16 Component Constraints Table

Used in the *Determine Internal Architecture Constraints* task in Section 7.2.2 to produce the INTERNAL ARCHITECTURE CONSTRAINTS workproduct.

Feature Ensembles		Architecture Variability Criteria			
		Separately Selectable/ Stand-alone	Masking	Internal Optimization	...
Partition 1 ①	Feature Ensemble 1.1	②	②	②	
	Feature Ensemble 1.2	H/L	L/L	L/H	
	...				
	Feature Ensemble 1.n				
Partition 2	Feature Ensemble 2.1				
	Feature Ensemble 2.2				
	...				
	Feature Ensemble 2.n				
...	...				
Partition n	Feature Ensemble n.n				
	Feature Ensemble n.n				
	...				
	Feature Ensemble n.n				

Exhibit C-16. Component Constraints Table

Key:

① Group FEs that partition ASSET BASE MODEL
② For each criterion, rate utility and feasibility

    L - Low
    - - Neutral
    H - High

C.2.17 Layering Constraints Table

Used in the *Determine Internal Architecture Constraints* task in Section 7.2.2 to produce the INTERNAL ARCHITECTURE CONSTRAINTS workproduct.

Subsets/ Layered Architecture Variants	Feature Ensembles						Separate Selectability	Internal Optimization	...
	FE 1	FE 2		...		FE n			
Layer 1 ①	X②	X					③		
Layer 2	X		X	X			H/L	L/L	
Layer 3	X	X	X	X			H/H	H/L	
Layer 4	X	X		X					
Layer 5	X		X	X					
...						X			
Layer n	X	X	X	X	X	X			

Exhibit C-17. Layering Constraints Table

Key:

① Put candidates, layers or specializations in approximate "subset - first" order
② Indicate which feature ensembles are part of the layer
③ Rate each layering variant for utility and feasibility

    L - Low
    - - Neutral
    H - High

References

In the following, documents which pertain most directly to ODM and its application are denoted with an asterisk (*).

[ADER96a]* STARS. Army STARS Demonstration Project Experience Report. Unisys STARS Technical Report STARS-VC-A011R/003/02, STARS Technology Center, Arlington VA, April 1996.

[Bail92a] Bailin, S. KAPTUR: Knowledge Acquisition for Preservation of Tradeoffs and Underlying Rationales. CTA Inc., Rockville MD, 1992.

[Bailey94] Bailey, K. Typologies and Taxonomies: An Introduction to Classification Techniques. Sage University Paper, Series: Quantitative Applications in the Social Sciences, Series/Number 07-102, Sage Publications, 1994.

[Bloc81]   Block, P. *Flawless Consulting: A Guide to Getting Your Expertise Used.* Pfeiffer & Co., San Diego CA, 1981.

[Boeh92a] Boehm, B., W. Scherlis. "Megaprogramming." In *Proceedings of the DARPA Software Technology Conference*, Arlington VA, April 1992.

[Boji91]   Bojie, D. "The Storytelling Organization: A Study of Story Performance in a Office-supply Firm." *Administrative Science Quarterly*, Vol. 36, 1991, pp.106-26.

[Brow91]   Brown, J. S. "Research that Reinvents the Corporation." *Harvard Business Review*, March-April 1991.

[CFRP93a]STARS. STARS Conceptual Framework for Reuse Processes (CFRP), Vol. I: Definition, Version 3.0. Unisys STARS Technical Report STARS-VC-A018/001/00, STARS Technology Center, Arlington VA, October 1993.

[CFRP93b]STARS. STARS Conceptual Framework for Reuse Processes (CFRP), Vol. II: Application, Version 1.0. Unisys STARS Technical Report STARS-VC-A018/002/00, STARS Technology Center, Arlington VA, September 1993.

[Clem95a] Clements, P., P. Kogut. "Features of Architecture Description Languages." In *Proceedings of the 7th Annual Software Technology Conference*, Salt Lake City UT, April 1995.

[Coll91a] * Collins, P. "Toward a Reusable Domain Analysis." In *Proceedings of the 4th Annual Workshop on Software Reuse*, Herndon VA, November 1991.

[Crep95a] Creps, R., M.J. Davis, M. Simos, et al. "Using a Conceptual Framework for Reuse Processes as a Basis for Reuse Planning." In *Proceedings of the 7th Annual Software Technology Conference*, Salt Lake City UT, April 1995.

[DEGB95a]* Army STARS Demonstration Project. Domain Engineering Guidebook. U.S. Army CECOM Software Engineering Directorate, Ft. Monmouth NJ, 1995.

[DISA93a]DISA/CIM Software Reuse Program. Domain Analysis and Design Process, Version 1. Technical Report 1222-04-210/30.1, DISA Center for Information Management, Arlington VA, March 1993.

[Fore92a] Foreman, J. "STARS Mission." In *Proceedings of the DARPA Software Technology Conference*, Arlington VA, April 1992.

[Goma90a]Gomaa, H. A Domain Requirements Analysis and Specification Method. Technical Report, George Mason University, Fairfax VA, February 1990.

[Grei72] Greiner, L. "Evolutions and Revolutions as Organizations Grow." *Harvard Business Review*, July-August 1972.

[Gris93] Griss, M. "Towards Tools and Languages for Hybrid Domain-Specific Kits." In *Proceedings of the Sixth Annual Workshop on Software Reuse*, Owego NY, November 1993.

[Hadd94] Haddock, G., and K. Harbison. "From Scenarios to Domain Models; Processes and Representations," in *Proceedings of Knowledge-based Artificial Intelligence Systems in Aerospace and Industry. SPIE: International Society for Optical Engineering*, April 1994.

[Henn96] Henninger, S. "Accelerating the Successful Reuse of Problem Solving Knowledge Through the Domain Lifecycle." In *Proceedings of the Fourth International Conference on Software Reuse*, IEEE Computer Society Press.

[IDEF81] Softech, Inc. "Integrated Computer-Aided Manufacturing (ICAM) Architecture Part II. Volume IV - Function Modeling Manual (IDEF0)." Technical Report AFWAL-TR-81-4023 Volume IV, Materials Laboratory (AFWAL/MLTC), AF Wright Aeronautical Laboratories (AFSC), Wright-Paterson AFB, Dayton OH, June 1981.

[Kang90a] Kang, K, S. Cohen, et al. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Pittsburgh PA, November 1990.

[Kell96] * Kelly, T.P., W. Lam, B.R. Whittle. "Diary of a domain analyst: a domain analysis case-study from avionics." In *Proceedings of IFIP Working Groups 8.1/13.2 Conference on Domain Knowledge for Interactive System Design*, Geneva, May 1996.

[Klin95a] Klingler, C. "A Practical Approach to Process Definition." In *Proceedings of the 7th Annual Software Technology Conference*, Salt Lake City UT, April 1995.

[LIBR96] STARS. Learning and Inquiry Based Reuse Adoption (LIBRA): A Field Guide to Reuse Adoption through Organizational Learning, Version 1.1. Loral Defense Systems-East STARS Technical Report STARS-PA33-AG01/001/02, STARS Technology Center, Arlington VA, February 1996.

[Liev80] Lievegoed, B. *The Developing Organization*. Celestial Arts, Millbrae CA, 1980.

[Lind93] Linde, C. "Reflections on Workplace Learning." Working Paper, Institute for Research on Learning, 2550 Hanover St., Palo Alto CA, 1993.

[Manc88] Mancuso, J.C., and M.L.C. Shaw. *Cognition and Personal Struture: Computer Access and Analysis*. Praeger Press, New York NY, 1988.

[Marc88] Marca, D., and C. McGowan. *SADT, Structured Analysis and Design Technique*. McGraw-Hill, New York NY, 1988.

[McNi88a] McNicholl, D., S. Cohen, et al. Common Ada Missile Packages - Phase 2 (CAMP-2) - Vol. 1: CAMP Parts and Parts Composition System. Final Report AFAL-TR-88-62, McDonnell Douglas, St. Louis MO, November 1988.

[Moor91a] Moore, G. *Crossing the Chasm: Marketing and Selling Technology Products to Mainstream Customers*. Harper Business, New York NY, 1991.

[Neig83a] Neighbors, J. "The Draco Approach to Constructing Software from Reusable Components." In *Proceedings of the Workshop on Reusability in Programming*. ITT Programming, Stratford CT, September 1983.

[Perr92a] Perry, D., A. Wolf. "Foundations for the Study of Software Architecture." *ACM Software Engineering Notes*, Vol. 17, #4, October 1992.

[Pine93] Pine, J. *Mass Customization: The New Frontier in Business Competition*. Harvard Business School Press, Boston MA, 1993.

[Prah89] Prahalad, C., and G. Hamel. "Strategic Intent." *Harvard Business Review*, May-June 1989.

[Prah90] Prahalad, C., and G. Hamel. "The Core Competence of the Corporation." *Harvard Business Review*, May-June 1990.

[Prie91a] Prieto-Diaz, R. Reuse Library Process Model. IBM STARS Technical Report CDRL 03041-002, STARS Technology Center, Arlington VA, July 1991.

[Prie91b] Prieto-Diaz, R., G. Arango. "Domain Analysis Concepts and Research Directions." In *Domain Analysis and Software Systems Modeling*, R. Prieto-Diaz and G. Arango, ed., IEEE Computer Society Press, 1991.

[ROSE93a] STARS. The Reuse-Oriented Software Evolution (ROSE) Process Model, Version 0.5. Unisys STARS Technical Report STARS-UC-05155/001/00, STARS Technology Center, Arlington VA, July 1993.

[RSM93a] STARS. Reuse Strategy Model: Planning Aid for Reuse-Based Projects. Boeing STARS Technical Report D613-55159, STARS Technology Center, Arlington VA, July 1993

[Sche87a] Schein, E. *The Clinical Perspective in Fieldwork*. Qualitative Research Methods Series 5, Sage, Newbury Park CA, 1987.

[Sche87b] Schein, E. *Process Consultation, Vol. 2: Lessons for Managers and Consultants*. Addison-Wesley, Reading MA, 1987.

[Schw91] Schwartz, P. *The Art of the Long View, Planning for the Future in an Uncertain World*. Doubleday/Currency, New York NY, 1991.

[Schw93] Schwartzman, H. *Ethnography in Organizations*. Qualitative Research Methods Series 27, Sage, Newbury Park CA, 1993.

[Seng90] Senge, P. *The Fifth Discipline*. Doubleday/Currency, New York NY, 1990.

[Seng94] Senge, P., et al. *The Fifth Discipline Fieldbook: Strategies and Tools for Building a Learning Organization*. Doubleday/Currency, New York NY, 1994.

[Shaw94a] Shaw, M. "Making Choices: A Comparison of Styles for Software Architecture."
Unpublished manuscript, Carnegie-Mellon University, Pittsburgh PA, May 1994.

[Shaw95a] Shaw, M, et al. "Abstractions for Software Architectures and Tools to Support Them."
IEEE Transactions on Software Engineering, 1995.

[Shaw96] Shaw, M, D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline.*
Prentice Hall, Upper Saddle River NJ, 1996.

[Shri90] Shrivastva, S. and D. Cooperrider. *Appreciative Management and Leadership.* Jossey-
Bass, San Francisco CA, 1990.

[Simo91a] * Simos, M. "The Growing of an Organon: A Hybrid Knowledge-Based Technology
and Methodology for Software Reuse." In *Domain Analysis and Software Systems
Modeling*, R. Prieto-Diaz and G. Arango, ed., IEEE Computer Society Press, 1991.

[Simo91b] * Simos, M., "Navigating Through Soundspace: Modeling the Sound Domain At Real
World." In *Proceedings of the 4th Annual Workshop on Software Reuse*, Herndon VA,
November 1991.

[Simo94a] * Simos, M., "Juggling in Free Fall: Uncertainty Management Aspects of Domain
Analysis Methods." In *Proceedings of the 5th International Conference on
Information Processing and Management of Uncertainty in Knowledge-Based
Systems*, Springer-Verlag, July 1994.

[Sold89a] Solderitsch, J., K. Wallnau, J. Thalhamer. "Constructing Domain-Specific Ada Reuse
Libraries." In *Proceedings of the 7th Annual National Conference on Ada Technology*,
March 1989.

[SPC93a] Software Productivity Consortium. Reuse Adoption Guidebook, Version 02.00.05.
Software Productivity Consortium, Herndon, VA, November, 1993.

[SPC93b] Software Productivity Consortium. Reuse-driven Software Processes Guidebook,
Version 02.00.03. Software Productivity Consortium, Herndon, VA, November, 1993.

[Spra79a] Spradley, J. *The Ethnographic Interview.* Holt, Rinehart, and Winston, New York NY,
1979.

[Spra79b] Spradley, J. *Ethnographic Observation.* Holt, Rinehart, and Winston, New York NY,
1979.

[Tris83] Trist, E. "Referent Organizations and the Development of Interorganizational
Domains." *Human Relations*, Vol. 36, No. 13, pp. 269-84, 1983.

[Unis88a] * Unisys. Reusability Library Framework AdaKNET and AdaTAU Design Report.
PAO D4705-CV-880601-1, Unisys Defense Systems, System Development Group,
Paoli PA, 1988.

[Wart92a] Wartik, S., R. Prieto-Diaz. "Criteria for Comparing Domain Analysis Approaches."
International Journal of Software Engineering and Knowledge Engineering,
September 1992.

[Weis92] Weisbord, M. *Discovering Common Ground.* Berrett-Koehler, San Francisco CA,
1992.

[Wick94a] * Wickman, G., J. Solderitsch, M. Simos. "A Systematic Software Reuse Program Based on an Architecture-Centric Domain Analysis." In *Proceedings of the 6th Annual Software Technology Conference*, Salt Lake City UT, April 1994.

[Wino87] Winograd, T., F. Flores. *Understanding Computers and Cognition: A New Foundation for Design*. Addison-Wesley, Reading MA, 1987.

[Zubo88] Zuboff, S. *In the Age of the Smart Machine*. Basic Books, New York NY, 1988.